Linux Device Drivers

Linux Device Drivers

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

Table of Contents

1. Driver Basics 1
1.1. Driver Entry and Exit points 1
module_init 1
module_exit 1
1.2. Atomic and pointer manipulation 2
atomic_read 2
atomic_set 3
atomic_add 4
atomic_sub 5
atomic_sub_and_test 6
atomic_inc 7
atomic_dec 7
atomic_dec_and_test 8
atomic_inc_and_test 9
atomic_add_negative 10
atomic_add_return 11
atomic_sub_return 11
atomic_add_unless 12
atomic_inc_short 13
atomic_or_long 14
1.3. Delaying, scheduling, and timer routines 15
struct task_cputime 15
struct thread_group_cputimer 16
pid_alive 17
is_global_init 18
wake_up_process 19
preempt_notifier_register 19
preempt_notifier_unregister 20
__wake_up 21
__wake_up_sync_key 22
complete 23
complete_all 24
wait_for_completion 25
wait_for_completion_timeout 26
wait_for_completion_interruptible 27
wait_for_completion_interruptible_timeout 27
wait_for_completion_killable 28
wait_for_completion_killable_timeout 29
try_wait_for_completion 30
completion_done 31
task_nice 32

sched_setscheduler 32
yield 33
yield_to 34
DECLARE_COMPLETION 35
DECLARE_COMPLETION_ONSTACK 36
init_completion 37
reinit_completion 38
INIT_COMPLETION 39
__round_jiffies 39
__round_jiffies_relative 41
round_jiffies 42
round_jiffies_relative 43
__round_jiffies_up 44
__round_jiffies_up_relative 44
round_jiffies_up 45
round_jiffies_up_relative 46
set_timer_slack 47
init_timer_key 48
mod_timer_pending 49
mod_timer 50
mod_timer_pinned 51
add_timer 52
add_timer_on 53
del_timer 54
try_to_del_timer_sync 55
del_timer_sync 56
schedule_timeout 57
msleep 58
msleep_interruptible 59
usleep_range 60
1.4. Wait queues and Wake events 60
wait_event 60
wait_event_timeout 61
wait_event_interruptible 62
wait_event_interruptible_timeout 63
wait_event_interruptible_locked 65
wait_event_interruptible_locked_irq 66
wait_event_interruptible_exclusive_locked 67
wait_event_interruptible_exclusive_locked_irq 68
wait_event_interruptible_lock_irq_timeout 69
wait_event_killable 71
wait_on_bit 72
wait_on_bit_lock 73

wait_on_atomic_t 74
finish_wait 75
abort_exclusive_wait 76
wake_up_bit 77
wake_up_atomic_t 78
1.5. High-resolution timers 79
ktime_set 79
ktime_sub 80
ktime_add 81
timespec_to_ktime 82
timeval_to_ktime 83
ktime_to_timespec 83
ktime_to_timeval 84
ktime_to_ns 85
ktime_equal 86
struct hrtimer 86
struct hrtimer_sleeper 88
struct hrtimer_clock_base 89
ktime_add_ns 90
ktime_sub_ns 91
hrtimer_forward 92
hrtimer_start_range_ns 93
hrtimer_start 94
hrtimer_try_to_cancel 95
hrtimer_cancel 96
hrtimer_get_remaining 96
hrtimer_init 97
hrtimer_get_res 98
schedule_hrtimeout_range 99
schedule_hrtimeout 100
1.6. Workqueues and Kevents 101
queue_work 102
queue_work_on 102
queue_delayed_work 103
queue_delayed_work_on 104
flush_workqueue 105
drain_workqueue 106
flush_work 107
flush_work_sync 108
cancel_work_sync 109
flush_delayed_work 110
flush_delayed_work_sync 111
cancel_delayed_work_sync 112

schedule_work 113
schedule_delayed_work 114
schedule_delayed_work_on 115
flush_scheduled_work 116
execute_in_process_context 117
destroy_workqueue 118
workqueue_set_max_active 119
workqueue_congested 120
work_cpu 121
work_busy 122
work_on_cpu 122
1.7. Internal Functions 123
reparent_to_kthreadd 124
wait_task_stopped 124
task_clear_group_stop_trapping 126
task_clear_group_stop_pending 127
task_participate_group_stop 128
do_notify_parent_cldstop 129
sys_restart_syscall 130
set_current_blocked 130
sys_rt_sigprocmask 131
sys_rt_sigpending 132
do_sigtimedwait 133
sys_rt_sigtimedwait 134
sys_kill 135
sys_tgkill 136
sys_tkill 137
sys_rt_sigqueueinfo 138
sys_sigpending 138
sys_sigprocmask 139
sys_rt_sigaction 140
sys_rt_sigsuspend 141
kthread_run 142
kthread_should_stop 143
kthread_create_on_node 144
kthread_bind 145
kthread_stop 146
kthread_worker_fn 147
queue_kthread_work 148
flush_kthread_work 149
flush_kthread_worker 149
1.8. Kernel objects manipulation 150
kobject_get_path 150

kobject_set_name 151
kobject_init 152
kobject_add 153
kobject_init_and_add 155
kobject_rename 156
kobject_del 157
kobject_get 157
kobject_put 158
kobject_create_and_add 159
kset_register 160
kset_unregister 161
kset_create_and_add 161
1.9. Kernel utility functions 162
upper_32_bits 163
lower_32_bits 163
might_sleep 164
trace_printk 165
min_not_zero 166
clamp 167
clamp_t 167
clamp_val 168
container_of 169
BUILD_BUG_ON 170
printk 171
console_lock 173
console_trylock 173
console_conditional_schedule 174
printk_timed_ratelimit 175
kmsg_dump_register 176
kmsg_dump_unregister 177
panic 177
emergency_restart 178
kernel_restart 179
kernel_halt 180
kernel_power_off 181
orderly_poweroff 182
rcu_read_lock_bh_held 182
init_rcu_head_on_stack 183
destroy_rcu_head_on_stack 184
1.10. Device Resource Management 185
devres_alloc 185
devres_for_each_res 186
devres_free 188

devres_add 189
devres_find 189
devres_get 191
devres_remove 192
devres_destroy 193
devres_open_group 194
devres_close_group 195
devres_remove_group 196
devres_release_group 197
devm_kzalloc 198
devm_kfree 199
2. Device drivers infrastructure 201
2.1. The Basic Device Driver-Model Structures 201
struct bus_type 201
struct device_driver 203
struct class 205
struct device 207
module_driver 211
2.2. Device Drivers Base 212
driver_for_each_device 213
driver_find_device 214
driver_create_file 215
driver_remove_file 215
driver_add_kobj 216
get_driver 217
put_driver 218
driver_register 219
driver_unregister 219
driver_find 220
dev_driver_string 221
device_create_file 222
device_remove_file 223
device_create_bin_file 223
device_remove_bin_file 224
device_schedule_callback_owner 225
device_initialize 226
dev_set_name 227
device_add 228
device_register 229
get_device 230
put_device 231
device_del 232
device_unregister 233

device_for_each_child 233
device_find_child 234
__root_device_register 236
root_device_unregister 237
device_create_vargs 237
device_create 239
device_destroy 240
device_rename 241
device_move 243
__class_create 244
class_destroy 245
class_dev_iter_init 246
class_dev_iter_next 247
class_dev_iter_exit 248
class_for_each_device 249
class_find_device 250
class_compat_register 251
class_compat_unregister 252
class_compat_create_link 252
class_compat_remove_link 253
request_firmware 254
release_firmware 255
request_firmware_nowait 256
cache_firmware 257
uncache_firmware 258
transport_class_register 259
transport_class_unregister 260
anon_transport_class_register 261
anon_transport_class_unregister 262
transport_setup_device 263
transport_add_device 263
transport_configure_device 264
transport_remove_device 265
transport_destroy_device 266
sysdev_driver_register 267
sysdev_driver_unregister 268
sysdev_register 269
platform_device_register_simple 269
platform_device_register_data 270
platform_get_resource 272
platform_get_irq 273
platform_get_resource_byname 273
platform_get_irq_byname 274

platform_add_devices 275
platform_device_put 276
platform_device_alloc 277
platform_device_add_resources 277
platform_device_add_data 278
platform_device_add 279
platform_device_del 280
platform_device_register 281
platform_device_unregister 282
platform_device_register_resndata 283
platform_driver_register 284
platform_driver_unregister 285
platform_driver_probe 285
platform_create_bundle 286
bus_for_each_dev 288
bus_find_device 289
bus_find_device_by_name 290
bus_for_each_drv 291
bus_rescan_devices 292
device_reprobe 293
bus_register 294
bus_unregister 295
2.3. Device Drivers Power Management 296
dpm_resume_noirq 296
dpm_resume_end 297
dpm_suspend_noirq 298
dpm_suspend_start 298
device_pm_wait_for_dev 299
dpm_for_each_dev 300
2.4. Device Drivers ACPI Support 301
acpi_bus_register_driver 301
acpi_bus_unregister_driver 302
acpi_bus_driver_init 303
2.5. Device drivers PnP support 304
pnp_register_protocol 304
pnp_unregister_protocol 305
pnp_request_card_device 305
pnp_release_card_device 306
pnp_register_card_driver 307
pnp_unregister_card_driver 307
pnp_add_id 308
pnp_start_dev 309
pnp_stop_dev 310

pnp_activate_dev 310
pnp_disable_dev 311
pnp_is_active 312
2.6. Userspace IO devices 313
uio_event_notify 313
__uio_register_device 313
uio_unregister_device 314
struct uio_mem 315
struct uio_port 316
struct uio_info 317
3. Parallel Port Devices 321
parport_yield 321
parport_yield_blocking 322
parport_wait_event 322
parport_wait_peripheral 323
parport_negotiate 325
parport_write 326
parport_read 327
parport_set_timeout 328
parport_register_driver 329
parport_unregister_driver 330
parport_get_port 331
parport_put_port 332
parport_register_port 333
parport_announce_port 334
parport_remove_port 335
parport_register_device 336
parport_unregister_device 338
parport_find_number 339
parport_find_base 340
parport_claim 340
parport_claim_or_block 341
parport_release 342
parport_open 343
parport_close 344
4. Message-based devices 347
4.1. Fusion message devices 347
mpt_register 347
mpt_deregister 348
mpt_event_register 349
mpt_event_deregister 350
mpt_reset_register 350

mpt_reset_deregister 351
mpt_device_driver_register 352
mpt_device_driver_deregister 353
mpt_get_msg_frame 354
mpt_put_msg_frame 355
mpt_put_msg_frame_hi_pri 356
mpt_free_msg_frame 357
mpt_send_handshake_request 358
mpt_verify_adapter 359
mpt_attach 360
mpt_detach 361
mpt_suspend 362
mpt_resume 362
mpt_GetIocState 363
mpt_alloc_fw_memory 364
mpt_free_fw_memory 365
mptbase_sas_persist_operation 366
mpt_raid_phys_disk_pg0 367
mpt_raid_phys_disk_get_num_paths 368
mpt_raid_phys_disk_pg1 369
mpt_findImVolumes 369
mpt_config 370
mpt_print_ioc_summary 371
mpt_set_taskmgmt_in_progress_flag 372
mpt_clear_taskmgmt_in_progress_flag 373
mpt_halt_firmware 374
mpt_Soft_Hard_ResetHandler 375
mpt_HardResetHandler 376
mpt_set_debug_level 377
mpt_get_cb_idx 378
mpt_is_discovery_complete 378
mpt_remove_dead_ioc_func 379
mpt_fault_reset_work 380
mpt_interrupt 381
mptbase_reply 382
mpt_add_sge 383
mpt_add_sge_64bit 384
mpt_add_sge_64bit_1078 385
mpt_add_chain 385
mpt_add_chain_64bit 386
mpt_host_page_access_control 387
mpt_host_page_alloc 388
mpt_get_product_name 389

convert_to_kb 390
mpt_do_ioc_recovery 391
mpt_detect_bound_ports 392
mpt_adapter_disable 393
mpt_adapter_dispose 394
MptDisplayIocCapabilities 395
MakeIocReady 395
GetIocFacts 396
GetPortFacts 397
SendIocInit 398
SendPortEnable 399
mpt_do_upload 400
mpt_downloadboot 401
KickStart 402
mpt_diag_reset 403
SendIocReset 405
initChainBuffers 406
PrimeIocFifos 406
mpt_handshake_req_reply_wait 407
WaitForDoorbellAck 409
WaitForDoorbellInt 410
WaitForDoorbellReply 411
GetLanConfigPages 412
GetIoUnitPage2 412
mpt_GetScsiPortSettings 413
mpt_readScsiDevicePageHeaders 414
mpt_inactive_raid_list_free 415
mpt_inactive_raid_volumes 416
mpt_sort_ioc_pg2 417
SendEventNotification 418
SendEventAck 419
mpt_ioc_reset 419
procmpt_create 420
procmpt_destroy 421
procmpt_summary_read 422
procmpt_version_read 423
procmpt_iocinfo_read 424
mpt_SoftResetHandler 425
ProcessEventNotification 426
mpt_fc_log_info 427
mpt_spi_log_info 428
mpt_sas_log_info 429
mpt_iocstatus_info_config 430

mpt_iocstatus_info 431
fusion_init 432
fusion_exit 433
mptscsih_info 434
mptscsih_proc_info 434
mptscsih_qcmd 436
mptscsih_IssueTaskMgmt 436
mptscsih_abort 438
mptscsih_dev_reset 439
mptscsih_bus_reset 439
mptscsih_host_reset 440
mptscsih_taskmgmt_complete 441
mptscsih_is_phys_disk 442
mptscsih_get_scsi_lookup 443
mptscsih_do_cmd 444
mptscsih_info_scsiio 445
_scsih_setup_eedp 446
_scsih_read_capacity_16 447
mptscsih_getclear_scsi_lookup 448
mptscsih_set_scsi_lookup 449
SCPNT_TO_LOOKUP_IDX 450
mptscsih_get_completion_code 451
mptscsih_synchronize_cache 452
mptscsih_device_sas_address_show 453
mptscsih_device_handle_show 454
mptscsih_device_fw_id_show 455
mptctl_syscall_down 456
mptspi_setTargetNegoParms 457
mptspi_writeIOCPage4 458
mptspi_initTarget 459
mptspi_is_raid 460
mptspi_print_write_nego 461
mptspi_print_read_nego 462
mptspi_dv_renegotiate_work 463
mptspi_ioc_reset 463
mptspi_resume 464
mptspi_probe 465
mptspi_init 466
mptspi_exit 466
mptfc_set_sdev_queue_depth 467
mptfc_init 468
mptfc_remove 469
mptfc_exit 469

lan_reply 470
4.2. I2O message devices 471
i2o_driver_notify_controller_add 471
i2o_driver_notify_controller_remove 472
i2o_driver_notify_device_add 473
i2o_driver_notify_device_remove 474
i2o_msg_out_to_virt 475
i2o_msg_in_to_virt 476
i2o_msg_get 477
i2o_msg_post 478
i2o_msg_post_wait 479
i2o_msg_nop_mfa 480
i2o_msg_nop 481
i2o_flush_reply 482
i2o_iop_free 483
i2o_msg_get_wait 483
i2o_cntxt_list_add 484
i2o_cntxt_list_remove 485
i2o_cntxt_list_get 486
i2o_cntxt_list_get_ptr 487
i2o_find_iop 488
i2o_iop_find_device 489
i2o_status_get 490
i2o_event_register 491
i2o_iop_quiesce 492
i2o_iop_enable 493
i2o_iop_quiesce_all 493
i2o_iop_enable_all 494
i2o_iop_clear 495
i2o_iop_init_outbound_queue 496
i2o_iop_reset 497
i2o_iop_activate 498
i2o_iop_systab_set 498
i2o_iop_online 499
i2o_iop_remove 500
i2o_systab_build 501
i2o_parse_hrt 502
i2o_iop_release 503
i2o_iop_alloc 503
i2o_iop_add 504
i2o_iop_init 505
i2o_iop_exit 506
i2o_config_init 507

i2o_config_exit 507
i2o_msg_post_wait_mem 508
i2o_exec_lct_get 510
i2o_exec_wait_alloc 510
i2o_exec_wait_free 511
i2o_msg_post_wait_complete 512
i2o_exec_show_vendor_id 513
i2o_exec_show_product_id 514
i2o_exec_probe 515
i2o_exec_remove 516
i2o_exec_lct_notify 517
i2o_exec_lct_modified 518
i2o_exec_reply 519
i2o_exec_event 520
i2o_exec_init 521
i2o_exec_exit 521
i2o_bus_scan 522
i2o_bus_store_scan 523
i2o_bus_probe 524
i2o_bus_remove 525
i2o_bus_init 526
i2o_bus_exit 526
i2o_device_claim 527
i2o_device_claim_release 528
i2o_device_issue_claim 529
i2o_device_release 530
i2o_device_show_class_id 531
i2o_device_show_tid 532
i2o_device_alloc 533
i2o_device_add 533
i2o_device_remove 534
i2o_device_parse_lct 535
i2o_bus_match 536
i2o_driver_dispatch 537
i2o_driver_init 538
i2o_driver_exit 539
i2o_pci_free 540
i2o_pci_alloc 540
i2o_pci_interrupt 541
i2o_pci_irq_enable 542
i2o_pci_irq_disable 543
i2o_pci_probe 544
i2o_pci_remove 545

i2o_pci_init 545
i2o_pci_exit 546
i2o_block_device_free 547
i2o_block_remove 548
i2o_block_device_flush 548
i2o_block_device_mount 549
i2o_block_device_lock 550
i2o_block_device_unlock 551
i2o_block_device_power 552
i2o_block_request_alloc 553
i2o_block_request_free 554
i2o_block_sglist_alloc 555
i2o_block_sglist_free 556
i2o_block_prep_req_fn 557
i2o_block_delayed_request_fn 558
i2o_block_end_request 558
i2o_block_reply 559
i2o_block_open 560
i2o_block_release 561
i2o_block_ioctl 562
i2o_block_check_events 563
i2o_block_transfer 564
i2o_block_request_fn 565
i2o_block_device_alloc 566
i2o_block_probe 567
i2o_block_init 568
i2o_block_exit 569
i2o_scsi_get_host 569
i2o_scsi_remove 570
i2o_scsi_probe 571
i2o_scsi_reply 572
i2o_scsi_notify_device_add 573
i2o_scsi_notify_device_remove 574
i2o_scsi_notify_controller_add 575
i2o_scsi_notify_controller_remove 575
i2o_scsi_queuecommand_lck 576
i2o_scsi_abort 577
i2o_scsi_bios_param 578
i2o_scsi_init 579
i2o_scsi_exit 580
i2o_get_class_name 581
i2o_proc_create_entries 582
i2o_proc_subdir_remove 583

i2o_proc_device_add 584
i2o_proc_iop_add 585
i2o_proc_iop_remove 586
i2o_proc_fs_create 586
i2o_proc_fs_destroy 587
i2o_proc_init 588
i2o_proc_exit 589
5. Sound Devices 591
snd_register_device 591
snd_printk 592
snd_printd 593
snd_BUG 594
snd_BUG_ON 594
snd_printdd 595
register_sound_special_device 596
register_sound_mixer 597
register_sound_midi 598
register_sound_dsp 599
unregister_sound_special 600
unregister_sound_mixer 601
unregister_sound_midi 602
unregister_sound_dsp 602
snd_pcm_playback_ready 603
snd_pcm_capture_ready 604
snd_pcm_playback_data 605
snd_pcm_playback_empty 606
snd_pcm_capture_empty 607
snd_pcm_format_cpu_endian 607
snd_pcm_lib_alloc_vmalloc_buffer 608
snd_pcm_lib_alloc_vmalloc_32_buffer 609
snd_pcm_new_stream 610
snd_pcm_new 611
snd_device_new 613
snd_device_free 614
snd_device_register 615
snd_iprintf 616
snd_info_get_line 617
snd_info_get_str 618
snd_info_create_module_entry 619
snd_info_create_card_entry 620
snd_card_proc_new 621
snd_info_free_entry 622
snd_info_register 623

snd_rawmidi_receive 623
snd_rawmidi_transmit_empty 624
snd_rawmidi_transmit_peek 625
snd_rawmidi_transmit_ack 626
snd_rawmidi_transmit 627
snd_rawmidi_new 628
snd_rawmidi_set_ops 630
snd_request_card 631
snd_lookup_minor_data 631
snd_register_device_for_dev 632
snd_unregister_device 634
copy_to_user_fromio 635
copy_from_user_toio 636
snd_pcm_lib_preallocate_free_for_all 637
snd_pcm_lib_preallocate_pages 638
snd_pcm_lib_preallocate_pages_for_all 639
snd_pcm_sgbuf_ops_page 640
snd_pcm_lib_malloc_pages 641
snd_pcm_lib_free_pages 642
snd_pcm_lib_free_vmalloc_buffer 643
snd_pcm_lib_get_vmalloc_page 644
snd_card_create 644
snd_card_disconnect 646
snd_card_unref 647
snd_card_set_id 647
snd_card_register 648
snd_component_add 649
snd_card_file_add 650
snd_card_file_remove 651
snd_power_wait 652
snd_dma_program 653
snd_dma_disable 654
snd_dma_pointer 655
snd_ctl_new1 656
snd_ctl_free_one 657
snd_ctl_add 658
snd_ctl_replace 659
snd_ctl_remove 660
snd_ctl_remove_id 661
snd_ctl_activate_id 662
snd_ctl_rename_id 662
snd_ctl_find_numid 663
snd_ctl_find_id 664

snd_ctl_enum_info 665
snd_pcm_set_ops 667
snd_pcm_set_sync 667
snd_interval_refine 668
snd_interval_ratnum 669
snd_interval_list 670
snd_pcm_hw_rule_add 671
snd_pcm_hw_constraint_integer 673
snd_pcm_hw_constraint_minmax 674
snd_pcm_hw_constraint_list 675
snd_pcm_hw_constraint_ratnums 676
snd_pcm_hw_constraint_ratdens 677
snd_pcm_hw_constraint_msbits 677
snd_pcm_hw_constraint_step 678
snd_pcm_hw_constraint_pow2 679
snd_pcm_hw_param_value 680
snd_pcm_hw_param_first 681
snd_pcm_hw_param_last 682
snd_pcm_lib_ioctl 684
snd_pcm_period_elapsed 684
snd_hwdep_new 685
snd_pcm_stop 686
snd_pcm_suspend 687
snd_pcm_suspend_all 688
snd_malloc_pages 689
snd_free_pages 690
snd_dma_alloc_pages 691
snd_dma_alloc_pages_fallback 692
snd_dma_free_pages 693
snd_dma_get_reserved_buf 694
snd_dma_reserve_buf 695
6. 16x50 UART Driver 697
/usr/src/linux-3.0.101-1//include/linux/serial_core.h 697
uart_update_timeout 697
uart_get_baud_rate 698
uart_get_divisor 699
uart_parse_options 700
uart_set_options 702
uart_register_driver 703
uart_unregister_driver 704
uart_add_one_port 704
uart_remove_one_port 705
uart_handle_dcd_change 706

uart_handle_cts_change 707
serial8250_suspend_port 708
serial8250_resume_port 709
serial8250_register_port 710
serial8250_unregister_port 710
7. Frame Buffer Library 713
7.1. Frame Buffer Memory 713
register_framebuffer 713
unregister_framebuffer 714
fb_set_suspend 715
fb_get_options 716
7.2. Frame Buffer Colormap 717
fb_dealloc_cmap 717
fb_copy_cmap 718
fb_set_cmap 718
fb_default_cmap 719
fb_invert_cmaps 720
7.3. Frame Buffer Video Mode Database 721
fb_try_mode 721
fb_delete_videomode 722
fb_find_mode 723
fb_var_to_videomode 725
fb_videomode_to_var 726
fb_mode_is_equal 726
fb_find_best_mode 727
fb_find_nearest_mode 728
fb_match_mode 729
fb_add_videomode 730
fb_destroy_modelist 731
fb_videomode_to_modelist 732
7.4. Frame Buffer Macintosh Video Mode Database 732
mac_vmode_to_var 733
mac_map_monitor_sense 734
mac_find_mode 734
7.5. Frame Buffer Fonts 736
8. Input Subsystem 737
8.1. Input core 737
struct input_absinfo 737
struct input_keymap_entry 738
struct ff_replay 739
struct ff_trigger 740
struct ff_envelope 741

struct ff_constant_effect 742
struct ff_ramp_effect 743
struct ff_condition_effect 744
struct ff_periodic_effect 745
struct ff_rumble_effect 746
struct ff_effect 747
struct input_dev 749
struct input_handler 754
struct input_handle 757
input_set_events_per_packet 758
struct ff_device 759
input_event 761
input_inject_event 762
input_alloc_absinfo 763
input_grab_device 764
input_release_device 765
input_open_device 766
input_close_device 766
input_scancode_to_scalar 767
input_get_keycode 768
input_set_keycode 769
input_reset_device 770
input_allocate_device 771
input_free_device 772
input_set_capability 773
input_register_device 774
input_unregister_device 775
input_register_handler 776
input_unregister_handler 776
input_handler_for_each_handle 777
input_register_handle 778
input_unregister_handle 779
input_ff_upload 780
input_ff_erase 781
input_ff_event 782
input_ff_create 783
input_ff_destroy 783
input_ff_create_memless 784
8.2. Multitouch Library 785
struct input_mt_slot 785
input_mt_init_slots 786
input_mt_destroy_slots 787
input_mt_report_slot_state 788

input_mt_report_finger_count 789
input_mt_report_pointer_emulation 790
8.3. Polled input devices 791
struct input_polled_dev 791
input_allocate_polled_device 792
input_free_polled_device 793
input_register_polled_device 794
input_unregister_polled_device 795
8.4. Matrix keyboars/keypads 796
struct matrix_keymap_data 796
struct matrix_keypad_platform_data 797
matrix_keypad_build_keymap 799
8.5. Sparse keymap support 800
struct key_entry 800
sparse_keymap_entry_from_scancode 801
sparse_keymap_entry_from_keycode 802
sparse_keymap_setup 803
sparse_keymap_free 804
sparse_keymap_report_entry 805
sparse_keymap_report_event 806
9. Serial Peripheral Interface (SPI) 809
struct spi_device 809
struct spi_driver 812
spi_unregister_driver 813
struct spi_master 814
struct spi_transfer 816
struct spi_message 819
spi_write 821
spi_read 822
spi_w8r8 823
spi_w8r16 824
struct spi_board_info 825
spi_register_board_info 827
spi_register_driver 828
spi_alloc_device 829
spi_add_device 830
spi_new_device 831
spi_alloc_master 832
spi_register_master 833
spi_unregister_master 834
spi_busnum_to_master 835
spi_setup 836
spi_async 837

spi_async_locked 839
spi_sync 840
spi_sync_locked 841
spi_bus_lock 843
spi_bus_unlock 844
spi_write_then_read 845
10. I²C and SMBus Subsystem 847
struct i2c_driver 847
struct i2c_client 850
struct i2c_board_info 851
I2C_BOARD_INFO 853
struct i2c_msg 854
i2c_register_board_info 855
i2c_verify_client 857
i2c_lock_adapter 858
i2c_unlock_adapter 858
i2c_new_device 859
i2c_unregister_device 860
i2c_new_dummy 861
i2c_add_adapter 862
i2c_add_numbered_adapter 863
i2c_del_adapter 864
i2c_del_driver 865
i2c_use_client 866
i2c_release_client 867
i2c_transfer 868
i2c_master_send 869
i2c_master_recv 869
i2c_smbus_read_byte 870
i2c_smbus_write_byte 871
i2c_smbus_read_byte_data 872
i2c_smbus_write_byte_data 873
i2c_smbus_read_word_data 874
i2c_smbus_write_word_data 875
i2c_smbus_process_call 876
i2c_smbus_read_block_data 877
i2c_smbus_write_block_data 878
i2c_smbus_xfer 879

Chapter 1. Driver Basics

1.1. Driver Entry and Exit points

module_init

LINUX

Kernel Hackers ManualApril 2016

Name

`module_init` — driver initialization entry point

Synopsis

```
module_init ( x);
```

Arguments

x

function to be run at kernel boot time or module insertion

Description

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

module_exit

LINUX

Kernel Hackers ManualApril 2016

Name

`module_exit` — driver exit entry point

Synopsis

```
module_exit ( x);
```

Arguments

x

function to be run when driver is removed

Description

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

1.2. Atomic and pointer manipulation

atomic_read

LINUX

Kernel Hackers ManualApril 2016

Name

`atomic_read` — read atomic variable

Synopsis

```
int atomic_read (const atomic_t * v);
```

Arguments

v

pointer of type atomic_t

Description

Atomically reads the value of *v*.

atomic_set

LINUX

Kernel Hackers ManualApril 2016

Name

`atomic_set` — set atomic variable

Synopsis

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

Arguments

v

pointer of type atomic_t

i

required value

Description

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

atomic_add

LINUX

Kernel Hackers ManualApril 2016

Name

atomic_add — add integer to atomic variable

Synopsis

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

Arguments

i

integer value to add

v

pointer of type atomic_t

Description

Atomically adds *i* to *v*.

atomic_sub

LINUX

Kernel Hackers ManualApril 2016

Name

atomic_sub — subtract integer from atomic variable

Synopsis

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

Arguments

i

integer value to subtract

v

pointer of type atomic_t

Description

Atomically subtracts *i* from *v*.

atomic_sub_and_test

LINUX

Kernel Hackers ManualApril 2016

Name

atomic_sub_and_test — subtract value from variable and test result

Synopsis

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

Arguments

i

integer value to subtract

v

pointer of type atomic_t

Description

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

atomic_inc

LINUX

Kernel Hackers ManualApril 2016

Name

`atomic_inc` — increment atomic variable

Synopsis

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

Arguments

`v`

pointer of type atomic_t

Description

Atomically increments `v` by 1.

atomic_dec

LINUX

Kernel Hackers ManualApril 2016

Name

`atomic_dec` — decrement atomic variable

Synopsis

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

Arguments

v

pointer of type atomic_t

Description

Atomically decrements *v* by 1.

atomic_dec_and_test

LINUX

Kernel Hackers ManualApril 2016

Name

`atomic_dec_and_test` — decrement and test

Synopsis

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

Arguments

v

pointer of type atomic_t

Description

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

atomic_inc_and_test

LINUX

Kernel Hackers ManualApril 2016

Name

atomic_inc_and_test — increment and test

Synopsis

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

Arguments

v

pointer of type atomic_t

Description

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

atomic_add_negative

LINUX

Kernel Hackers ManualApril 2016

Name

`atomic_add_negative` — add and test if negative

Synopsis

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

Arguments

i

integer value to add

v

pointer of type atomic_t

Description

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

atomic_add_return

LINUX

Kernel Hackers ManualApril 2016

Name

`atomic_add_return` — add integer and return

Synopsis

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

Arguments

i

integer value to add

v

pointer of type atomic_t

Description

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

atomic_sub_return

LINUX

Kernel Hackers ManualApril 2016

Name

`atomic_sub_return` — subtract integer and return

Synopsis

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

Arguments

i

integer value to subtract

v

pointer of type atomic_t

Description

Atomically subtracts *i* from *v* and returns *v* - *i*

atomic_add_unless

LINUX

Name

`atomic_add_unless` — add unless the number is already a given value

Synopsis

```
int atomic_add_unless (atomic_t * v, int a, int u);
```

Arguments

v

pointer of type atomic_t

a

the amount to add to v...

u

...unless v is equal to u.

Description

Atomically adds *a* to *v*, so long as *v* was not already *u*. Returns non-zero if *v* was not *u*, and zero otherwise.

atomic_inc_short

LINUX

Name

`atomic_inc_short` — increment of a short integer

Synopsis

```
short int atomic_inc_short (short int * v);
```

Arguments

v

pointer to type int

Description

Atomically adds 1 to *v* Returns the new value of *u*

atomic_or_long

LINUX

Name

`atomic_or_long` — OR of two long integers

Synopsis

```
void atomic_or_long (unsigned long * v1, unsigned long v2);
```

Arguments

v1

pointer to type unsigned long

v2

pointer to type unsigned long

Description

Atomically ORs *v1* and *v2* Returns the result of the OR

1.3. Delaying, scheduling, and timer routines

struct task_cputime

LINUX

Kernel Hackers ManualApril 2016

Name

`struct task_cputime` — collected CPU time counts

Synopsis

```
struct task_cputime {
```

```
  cputime_t utime;
  cputime_t stime;
  unsigned long long sum_exec_runtime;
};
```

Members

utime

time spent in user mode, in cputime_t units

stime

time spent in kernel mode, in cputime_t units

sum_exec_runtime

total time spent on the CPU, in nanoseconds

Description

This structure groups together three kinds of CPU time that are tracked for threads and thread groups. Most things considering CPU time want to group these counts together and treat all three of them in parallel.

struct thread_group_cputimer

LINUX

Kernel Hackers ManualApril 2016

Name

`struct thread_group_cputimer` — thread group interval timer counts

Synopsis

```
struct thread_group_cputimer {
```

```
  struct task_cputime cputime;
  int running;
  spinlock_t lock;
};
```

Members

cputime

thread group interval timers.

running

non-zero when there are timers running and *cputime* receives updates.

lock

lock for fields in this struct.

Description

This structure contains the version of task_cputime, above, that is used for thread group CPU timer calculations.

pid_alive

LINUX

Kernel Hackers ManualApril 2016

Name

pid_alive — check that a task structure is not stale

Synopsis

```
int pid_alive (struct task_struct * p);
```

Arguments

p

Task structure to be checked.

Description

Test if a process is not yet dead (at most zombie state) If pid_alive fails, then pointers within the task structure can be stale and must not be dereferenced.

is_global_init

LINUX

Kernel Hackers ManualApril 2016

Name

`is_global_init` — check if a task structure is init

Synopsis

```
int is_global_init (struct task_struct * tsk);
```

Arguments

tsk

Task structure to be checked.

Description

Check if a task structure is the first user space task the kernel created.

wake_up_process

LINUX

Kernel Hackers ManualApril 2016

Name

`wake_up_process` — Wake up a specific process

Synopsis

```
int wake_up_process (struct task_struct * p);
```

Arguments

p

The process to be woken up.

Description

Attempt to wake up the nominated process and move it to the set of runnable processes. Returns 1 if the process was woken up, 0 if it was already running.

It may be assumed that this function implies a write memory barrier before changing the task state if and only if any tasks are woken up.

preempt_notifier_register

LINUX

Kernel Hackers ManualApril 2016

Name

`preempt_notifier_register` — tell me when current is being preempted & rescheduled

Synopsis

```
void preempt_notifier_register (struct preempt_notifier * notifier);
```

Arguments

`notifier`

notifier struct to register

preempt_notifier_unregister

LINUX

Kernel Hackers ManualApril 2016

Name

`preempt_notifier_unregister` — no longer interested in preemption notifications

Synopsis

```
void preempt_notifier_unregister (struct preempt_notifier * notifier);
```

Arguments

notifier

notifier struct to unregister

Description

This is safe to call from within a preemption notifier.

__wake_up

LINUX

Kernel Hackers ManualApril 2016

Name

__wake_up — wake up threads blocked on a waitqueue.

Synopsis

```
void __wake_up (wait_queue_head_t * q, unsigned int mode, int nr_exclusive, void * key);
```

Arguments

q

the waitqueue

mode

which threads

nr_exclusive

how many wake-one or wake-many threads to wake up

key

is directly passed to the wakeup function

Description

It may be assumed that this function implies a write memory barrier before changing the task state if and only if any tasks are woken up.

__wake_up_sync_key

LINUX

Kernel Hackers ManualApril 2016

Name

__wake_up_sync_key — wake up threads blocked on a waitqueue.

Synopsis

```
void __wake_up_sync_key (wait_queue_head_t * q, unsigned int
mode, int nr_exclusive, void * key);
```

Arguments

q

the waitqueue

mode

which threads

nr_exclusive

how many wake-one or wake-many threads to wake up

key

opaque value to be passed to wakeup targets

Description

The sync wakeup differs that the waker knows that it will schedule away soon, so while the target thread will be woken up, it will not be migrated to another CPU - ie. the two threads are 'synchronized' with each other. This can prevent needless bouncing between CPUs.

On UP it can prevent extra preemption.

It may be assumed that this function implies a write memory barrier before changing the task state if and only if any tasks are woken up.

complete

LINUX

Kernel Hackers ManualApril 2016

Name

`complete` — signals a single thread waiting on this completion

Synopsis

```
void complete (struct completion * x);
```

Arguments

x

holds the state of this particular completion

Description

This will wake up a single thread waiting on this completion. Threads will be awakened in the same order in which they were queued.

See also `complete_all`, `wait_for_completion` and related routines.

It may be assumed that this function implies a write memory barrier before changing the task state if and only if any tasks are woken up.

complete_all

LINUX

Kernel Hackers ManualApril 2016

Name

`complete_all` — signals all threads waiting on this completion

Synopsis

```
void complete_all (struct completion * x);
```

Arguments

x

holds the state of this particular completion

Description

This will wake up all threads waiting on this particular completion event.

It may be assumed that this function implies a write memory barrier before changing the task state if and only if any tasks are woken up.

wait_for_completion

LINUX

Kernel Hackers ManualApril 2016

Name

`wait_for_completion` — waits for completion of a task

Synopsis

```
void __sched wait_for_completion (struct completion * x);
```

Arguments

x

holds the state of this particular completion

Description

This waits to be signaled for completion of a specific task. It is NOT interruptible and there is no timeout.

See also similar routines (i.e. `wait_for_completion_timeout`) with timeout and interrupt capability. Also see `complete`.

wait_for_completion_timeout

LINUX

Kernel Hackers ManualApril 2016

Name

`wait_for_completion_timeout` — waits for completion of a task (w/timeout)

Synopsis

```
unsigned long __sched wait_for_completion_timeout (struct
completion * x, unsigned long timeout);
```

Arguments

x

holds the state of this particular completion

timeout

timeout value in jiffies

Description

This waits for either a completion of a specific task to be signaled or for a specified timeout to expire. The timeout is in jiffies. It is not interruptible.

wait_for_completion_interruptible

LINUX

Kernel Hackers ManualApril 2016

Name

`wait_for_completion_interruptible` — waits for completion of a task (w/intr)

Synopsis

```
int __sched wait_for_completion_interruptible (struct
completion * x);
```

Arguments

x

holds the state of this particular completion

Description

This waits for completion of a specific task to be signaled. It is interruptible.

wait_for_completion_interruptible_timeout

LINUX

Kernel Hackers ManualApril 2016

Name

`wait_for_completion_interruptible_timeout` — waits for completion (w/(to,intr))

Synopsis

```
long __sched wait_for_completion_interruptible_timeout (struct
completion * x, unsigned long timeout);
```

Arguments

x

holds the state of this particular completion

timeout

timeout value in jiffies

Description

This waits for either a completion of a specific task to be signaled or for a specified timeout to expire. It is interruptible. The timeout is in jiffies.

wait_for_completion_killable

LINUX

Name

`wait_for_completion_killable` — waits for completion of a task (killable)

Synopsis

```
int __sched wait_for_completion_killable (struct completion * x);
```

Arguments

x

holds the state of this particular completion

Description

This waits to be signaled for completion of a specific task. It can be interrupted by a kill signal.

wait_for_completion_killable_timeout

LINUX

Name

`wait_for_completion_killable_timeout` — waits for completion of a task (w/(to,killable))

Synopsis

```
long __sched wait_for_completion_killable_timeout (struct
completion * x, unsigned long timeout);
```

Arguments

x

holds the state of this particular completion

timeout

timeout value in jiffies

Description

This waits for either a completion of a specific task to be signaled or for a specified timeout to expire. It can be interrupted by a kill signal. The timeout is in jiffies.

try_wait_for_completion

LINUX

Kernel Hackers ManualApril 2016

Name

`try_wait_for_completion` — try to decrement a completion without blocking

Synopsis

```
bool try_wait_for_completion (struct completion * x);
```

Arguments

x

completion structure

Returns

0 if a decrement cannot be done without blocking 1 if a decrement succeeded.

If a completion is being used as a counting completion, attempt to decrement the counter without blocking. This enables us to avoid waiting if the resource the completion is protecting is not available.

completion_done

LINUX

Kernel Hackers ManualApril 2016

Name

`completion_done` — Test to see if a completion has any waiters

Synopsis

```
bool completion_done (struct completion * x);
```

Arguments

x

completion structure

Returns

0 if there are waiters (`wait_for_completion` in progress) 1 if there are no waiters.

task_nice

LINUX

Kernel Hackers ManualApril 2016

Name

`task_nice` — return the nice value of a given task.

Synopsis

```
int task_nice (const struct task_struct * p);
```

Arguments

p

the task in question.

sched_setscheduler

LINUX

Kernel Hackers ManualApril 2016

Name

`sched_setscheduler` — change the scheduling policy and/or RT priority of a thread.

Synopsis

```
int sched_setscheduler (struct task_struct * p, int policy,
const struct sched_param * param);
```

Arguments

p

the task in question.

policy

new policy.

param

structure containing the new RT priority.

Description

NOTE that the task may be already dead.

yield

LINUX

Kernel Hackers ManualApril 2016

Name

`yield` — yield the current processor to other threads.

Synopsis

```
void __sched yield ( void);
```

Arguments

void

no arguments

Description

This is a shortcut for kernel-space yielding - it marks the thread runnable and calls `sys_sched_yield`.

yield_to

LINUX

Name

`yield_to` — yield the current processor to another thread in your thread group, or accelerate that thread toward the processor it's on.

Synopsis

```
bool __sched yield_to (struct task_struct * p, bool preempt);
```

Arguments

p

target task

preempt

whether task preemption is allowed or not

Description

It's the caller's job to ensure that the target task struct can't go away on us before we can do any checks.

Returns true if we indeed boosted the target task.

DECLARE_COMPLETION

LINUX

Name

DECLARE_COMPLETION — declare and initialize a completion structure

Synopsis

```
DECLARE_COMPLETION ( work);
```

Arguments

work

identifier for the completion structure

Description

This macro declares and initializes a completion structure. Generally used for static declarations. You should use the _ONSTACK variant for automatic variables.

DECLARE_COMPLETION_ONSTACK

LINUX

Name

DECLARE_COMPLETION_ONSTACK — declare and initialize a completion structure

Synopsis

```
DECLARE_COMPLETION_ONSTACK ( work);
```

Arguments

work

identifier for the completion structure

Description

This macro declares and initializes a completion structure on the kernel stack.

init_completion

LINUX

Kernel Hackers ManualApril 2016

Name

`init_completion` — Initialize a dynamically allocated completion

Synopsis

```
void init_completion (struct completion * x);
```

Arguments

x

pointer to completion structure that is to be initialized

Description

This inline function will initialize a dynamically created completion structure.

reinit_completion

LINUX

Kernel Hackers ManualApril 2016

Name

`reinit_completion` — reinitialize a completion structure

Synopsis

```
void reinit_completion (struct completion * x);
```

Arguments

x

pointer to completion structure that is to be reinitialized

Description

This inline function should be used to reinitialize a completion structure so it can be reused. This is especially important after `complete_all` is used.

INIT_COMPLETION

LINUX

Kernel Hackers ManualApril 2016

Name

`INIT_COMPLETION` — reinitialize a completion structure

Synopsis

```
INIT_COMPLETION ( x);
```

Arguments

x

completion structure to be reinitialized

Description

This macro should be used to reinitialize a completion structure so it can be reused. This is especially important after `complete_all` is used.

__round_jiffies

LINUX

Kernel Hackers ManualApril 2016

Name

__round_jiffies — function to round jiffies to a full second

Synopsis

```
unsigned long __round_jiffies (unsigned long j, int cpu);
```

Arguments

j

the time in (absolute) jiffies that should be rounded

cpu

the processor number on which the timeout will happen

Description

__round_jiffies rounds an absolute time in the future (in jiffies) up or down to (approximately) full seconds. This is useful for timers for which the exact time they fire does not matter too much, as long as they fire approximately every X seconds.

By rounding these timers to whole seconds, all such timers will fire at the same time, rather than at various times spread out. The goal of this is to have the CPU wake up less, which saves power.

The exact rounding is skewed for each processor to avoid all processors firing at the exact same time, which could lead to lock contention or spurious cache line bouncing.

The return value is the rounded version of the *j* parameter.

__round_jiffies_relative

LINUX

Kernel Hackers ManualApril 2016

Name

__round_jiffies_relative — function to round jiffies to a full second

Synopsis

```
unsigned long __round_jiffies_relative (unsigned long j, int cpu);
```

Arguments

j

the time in (relative) jiffies that should be rounded

cpu

the processor number on which the timeout will happen

Description

__round_jiffies_relative rounds a time delta in the future (in jiffies) up or down to (approximately) full seconds. This is useful for timers for which the exact time they fire does not matter too much, as long as they fire approximately every X seconds.

By rounding these timers to whole seconds, all such timers will fire at the same time, rather than at various times spread out. The goal of this is to have the CPU wake up less, which saves power.

The exact rounding is skewed for each processor to avoid all processors firing at the exact same time, which could lead to lock contention or spurious cache line bouncing.

The return value is the rounded version of the `j` parameter.

round_jiffies

LINUX

Kernel Hackers ManualApril 2016

Name

`round_jiffies` — function to round jiffies to a full second

Synopsis

```
unsigned long round_jiffies (unsigned long j);
```

Arguments

`j`

the time in (absolute) jiffies that should be rounded

Description

`round_jiffies` rounds an absolute time in the future (in jiffies) up or down to (approximately) full seconds. This is useful for timers for which the exact time they fire does not matter too much, as long as they fire approximately every X seconds.

By rounding these timers to whole seconds, all such timers will fire at the same time, rather than at various times spread out. The goal of this is to have the CPU wake up less, which saves power.

The return value is the rounded version of the `j` parameter.

round_jiffies_relative

LINUX

Kernel Hackers ManualApril 2016

Name

`round_jiffies_relative` — function to round jiffies to a full second

Synopsis

```
unsigned long round_jiffies_relative (unsigned long j);
```

Arguments

`j`

the time in (relative) jiffies that should be rounded

Description

`round_jiffies_relative` rounds a time delta in the future (in jiffies) up or down to (approximately) full seconds. This is useful for timers for which the exact time they fire does not matter too much, as long as they fire approximately every X seconds.

By rounding these timers to whole seconds, all such timers will fire at the same time, rather than at various times spread out. The goal of this is to have the CPU wake up less, which saves power.

The return value is the rounded version of the `j` parameter.

__round_jiffies_up

LINUX

Kernel Hackers ManualApril 2016

Name

__round_jiffies_up — function to round jiffies up to a full second

Synopsis

```
unsigned long __round_jiffies_up (unsigned long j, int cpu);
```

Arguments

j

the time in (absolute) jiffies that should be rounded

cpu

the processor number on which the timeout will happen

Description

This is the same as __round_jiffies except that it will never round down. This is useful for timeouts for which the exact time of firing does not matter too much, as long as they don't fire too early.

__round_jiffies_up_relative

LINUX

Kernel Hackers ManualApril 2016

Name

__round_jiffies_up_relative — function to round jiffies up to a full second

Synopsis

```
unsigned long __round_jiffies_up_relative (unsigned long j,
int cpu);
```

Arguments

j

the time in (relative) jiffies that should be rounded

cpu

the processor number on which the timeout will happen

Description

This is the same as __round_jiffies_relative except that it will never round down. This is useful for timeouts for which the exact time of firing does not matter too much, as long as they don't fire too early.

round_jiffies_up

LINUX

Kernel Hackers ManualApril 2016

Name

`round_jiffies_up` — function to round jiffies up to a full second

Synopsis

```
unsigned long round_jiffies_up (unsigned long j);
```

Arguments

j

the time in (absolute) jiffies that should be rounded

Description

This is the same as `round_jiffies` except that it will never round down. This is useful for timeouts for which the exact time of firing does not matter too much, as long as they don't fire too early.

round_jiffies_up_relative

LINUX

Name

round_jiffies_up_relative — function to round jiffies up to a full second

Synopsis

```
unsigned long round_jiffies_up_relative (unsigned long j);
```

Arguments

j

the time in (relative) jiffies that should be rounded

Description

This is the same as round_jiffies_relative except that it will never round down. This is useful for timeouts for which the exact time of firing does not matter too much, as long as they don't fire too early.

set_timer_slack

LINUX

Name

set_timer_slack — set the allowed slack for a timer

Synopsis

```
void set_timer_slack (struct timer_list * timer, int
slack_hz);
```

Arguments

timer

the timer to be modified

slack_hz

the amount of time (in jiffies) allowed for rounding

Description

Set the amount of time, in jiffies, that a certain timer has in terms of slack. By setting this value, the timer subsystem will schedule the actual timer somewhere between the time `mod_timer` asks for, and that time plus the slack.

By setting the slack to -1, a percentage of the delay is used instead.

init_timer_key

LINUX

Kernel Hackers ManualApril 2016

Name

`init_timer_key` — initialize a timer

Synopsis

```
void init_timer_key (struct timer_list * timer, const char *
name, struct lock_class_key * key);
```

Arguments

timer

the timer to be initialized

name

name of the timer

key

lockdep class key of the fake lock used for tracking timer sync lock dependencies

Description

`init_timer_key` must be done to a timer prior calling *any* of the other timer functions.

mod_timer_pending

LINUX

Kernel Hackers ManualApril 2016

Name

`mod_timer_pending` — modify a pending timer's timeout

Synopsis

```
int mod_timer_pending (struct timer_list * timer, unsigned
long expires);
```

Arguments

timer

the pending timer to be modified

expires

new timeout in jiffies

Description

`mod_timer_pending` is the same for pending timers as `mod_timer`, but will not re-activate and modify already deleted timers.

It is useful for unserialized use of timers.

mod_timer

LINUX

Kernel Hackers ManualApril 2016

Name

`mod_timer` — modify a timer's timeout

Synopsis

```
int mod_timer (struct timer_list * timer, unsigned long
expires);
```

Arguments

timer

the timer to be modified

expires

new timeout in jiffies

Description

`mod_timer` is a more efficient way to update the expire field of an active timer (if the timer is inactive it will be activated)

mod_timer(timer, expires) is equivalent to:

del_timer(timer); timer->expires = expires; add_timer(timer);

Note that if there are multiple unserialized concurrent users of the same timer, then `mod_timer` is the only safe way to modify the timeout, since `add_timer` cannot modify an already running timer.

The function returns whether it has modified a pending timer or not. (ie. `mod_timer` of an inactive timer returns 0, `mod_timer` of an active timer returns 1.)

mod_timer_pinned

LINUX

Name

`mod_timer_pinned` — modify a timer's timeout

Synopsis

```
int mod_timer_pinned (struct timer_list * timer, unsigned long
expires);
```

Arguments

`timer`

the timer to be modified

`expires`

new timeout in jiffies

Description

`mod_timer_pinned` is a way to update the expire field of an active timer (if the timer is inactive it will be activated) and not allow the timer to be migrated to a different CPU.

mod_timer_pinned(timer, expires) is equivalent to:

del_timer(timer); timer->expires = expires; add_timer(timer);

add_timer

LINUX

Name

`add_timer` — start a timer

Synopsis

```
void add_timer (struct timer_list * timer);
```

Arguments

`timer`

the timer to be added

Description

The kernel will do a ->function(->data) callback from the timer interrupt at the ->expires point in the future. The current time is 'jiffies'.

The timer's ->expires, ->function (and if the handler uses it, ->data) fields must be set prior calling this function.

Timers with an ->expires field in the past will be executed in the next timer tick.

add_timer_on

LINUX

Name

`add_timer_on` — start a timer on a particular CPU

Synopsis

```
void add_timer_on (struct timer_list * timer, int cpu);
```

Arguments

timer

the timer to be added

cpu

the CPU to start it on

Description

This is not very scalable on SMP. Double adds are not possible.

del_timer

LINUX

Kernel Hackers ManualApril 2016

Name

del_timer — deactive a timer.

Synopsis

```
int del_timer (struct timer_list * timer);
```

Arguments

timer

the timer to be deactivated

Description

`del_timer` deactivates a timer - this works on both active and inactive timers.

The function returns whether it has deactivated a pending timer or not. (ie. `del_timer` of an inactive timer returns 0, `del_timer` of an active timer returns 1.)

try_to_del_timer_sync

LINUX

Kernel Hackers ManualApril 2016

Name

`try_to_del_timer_sync` — Try to deactivate a timer

Synopsis

```
int try_to_del_timer_sync (struct timer_list * timer);
```

Arguments

timer

timer do del

Description

This function tries to deactivate a timer. Upon successful (ret >= 0) exit the timer is not queued and the handler is not running on any CPU.

del_timer_sync

LINUX

Kernel Hackers ManualApril 2016

Name

`del_timer_sync` — deactivate a timer and wait for the handler to finish.

Synopsis

```
int del_timer_sync (struct timer_list * timer);
```

Arguments

`timer`

the timer to be deactivated

Description

This function only differs from `del_timer` on SMP: besides deactivating the timer it also makes sure the handler has finished executing on other CPUs.

Synchronization rules

Callers must prevent restarting of the timer, otherwise this function is meaningless. It must not be called from interrupt contexts. The caller must not hold locks which would prevent completion of the timer's handler. The timer's handler must not call `add_timer_on`. Upon exit the timer is not queued and the handler is not running on any CPU.

Note

You must not hold locks that are held in interrupt context while calling this function. Even if the lock has nothing to do with the timer in question. Here's why:

CPU0 CPU1 ---- ---- <SOFTIRQ> `call_timer_fn`; base->running_timer = mytimer; spin_lock_irq(somelock); <IRQ> spin_lock(somelock); del_timer_sync(mytimer); while (base->running_timer == mytimer);

Now `del_timer_sync` will never return and never release somelock. The interrupt on the other CPU is waiting to grab somelock but it has interrupted the softirq that CPU0 is waiting to finish.

The function returns whether it has deactivated a pending timer or not.

schedule_timeout

LINUX

Kernel Hackers ManualApril 2016

Name

`schedule_timeout` — sleep until timeout

Synopsis

```
signed long __sched schedule_timeout (signed long timeout);
```

Arguments

timeout

timeout value in jiffies

Description

Make the current task sleep until *timeout* jiffies have elapsed. The routine will return immediately unless the current task state has been set (see set_current_state).

You can set the task state as follows -

TASK_UNINTERRUPTIBLE - at least *timeout* jiffies are guaranteed to pass before the routine returns. The routine will return 0

TASK_INTERRUPTIBLE - the routine may return early if a signal is delivered to the current task. In this case the remaining time in jiffies will be returned, or 0 if the timer expired in time

The current task state is guaranteed to be TASK_RUNNING when this routine returns.

Specifying a *timeout* value of MAX_SCHEDULE_TIMEOUT will schedule the CPU away without a bound on the timeout. In this case the return value will be MAX_SCHEDULE_TIMEOUT.

In all cases the return value is guaranteed to be non-negative.

msleep

LINUX

Kernel Hackers ManualApril 2016

Name

msleep — sleep safely even with waitqueue interruptions

Synopsis

```
void msleep (unsigned int msecs);
```

Arguments

msecs

Time in milliseconds to sleep for

msleep_interruptible

LINUX

Kernel Hackers ManualApril 2016

Name

`msleep_interruptible` — sleep waiting for signals

Synopsis

```
unsigned long msleep_interruptible (unsigned int msecs);
```

Arguments

msecs

Time in milliseconds to sleep for

usleep_range

LINUX

Kernel Hackers ManualApril 2016

Name

`usleep_range` — Drop in replacement for udelay where wakeup is flexible

Synopsis

```
void usleep_range (unsigned long min, unsigned long max);
```

Arguments

min

Minimum time in usecs to sleep

max

Maximum time in usecs to sleep

1.4. Wait queues and Wake events

wait_event

LINUX

Name

`wait_event` — sleep until a condition gets true

Synopsis

```
wait_event ( wq,  condition);
```

Arguments

wq

the waitqueue to wait on

condition

a C expression for the event to wait for

Description

The process is put to sleep (TASK_UNINTERRUPTIBLE) until the *condition* evaluates to true. The *condition* is checked each time the waitqueue *wq* is woken up.

`wake_up` has to be called after changing any variable that could change the result of the wait condition.

wait_event_timeout

LINUX

Name

`wait_event_timeout` — sleep until a condition gets true or a timeout elapses

Synopsis

```
wait_event_timeout ( wq,  condition,  timeout);
```

Arguments

`wq`

the waitqueue to wait on

`condition`

a C expression for the event to wait for

`timeout`

timeout, in jiffies

Description

The process is put to sleep (TASK_UNINTERRUPTIBLE) until the *`condition`* evaluates to true. The *`condition`* is checked each time the waitqueue *`wq`* is woken up.

`wake_up` has to be called after changing any variable that could change the result of the wait condition.

The function returns 0 if the *`timeout`* elapsed, or the remaining jiffies (at least 1) if the *`condition`* evaluated to `true` before the *`timeout`* elapsed.

wait_event_interruptible

LINUX

Kernel Hackers ManualApril 2016

Name

`wait_event_interruptible` — sleep until a condition gets true

Synopsis

```
wait_event_interruptible ( wq,  condition);
```

Arguments

wq

the waitqueue to wait on

condition

a C expression for the event to wait for

Description

The process is put to sleep (TASK_INTERRUPTIBLE) until the *condition* evaluates to true or a signal is received. The *condition* is checked each time the waitqueue *wq* is woken up.

`wake_up` has to be called after changing any variable that could change the result of the wait condition.

The function will return -ERESTARTSYS if it was interrupted by a signal and 0 if *condition* evaluated to true.

wait_event_interruptible_timeout

LINUX

Kernel Hackers ManualApril 2016

Name

`wait_event_interruptible_timeout` — sleep until a condition gets true or a timeout elapses

Synopsis

```
wait_event_interruptible_timeout ( wq,  condition,  timeout);
```

Arguments

`wq`

the waitqueue to wait on

`condition`

a C expression for the event to wait for

`timeout`

timeout, in jiffies

Description

The process is put to sleep (TASK_INTERRUPTIBLE) until the *`condition`* evaluates to true or a signal is received. The *`condition`* is checked each time the waitqueue *`wq`* is woken up.

`wake_up` has to be called after changing any variable that could change the result of the wait condition.

Returns

0 if the *timeout* elapsed, -ERESTARTSYS if it was interrupted by a signal, or the remaining jiffies (at least 1) if the *condition* evaluated to true before the *timeout* elapsed.

wait_event_interruptible_locked

LINUX

Kernel Hackers ManualApril 2016

Name

wait_event_interruptible_locked — sleep until a condition gets true

Synopsis

```
wait_event_interruptible_locked ( wq,  condition);
```

Arguments

wq

the waitqueue to wait on

condition

a C expression for the event to wait for

Description

The process is put to sleep (TASK_INTERRUPTIBLE) until the *condition* evaluates to true or a signal is received. The *condition* is checked each time the waitqueue *wq* is woken up.

It must be called with wq.lock being held. This spinlock is unlocked while sleeping but *condition* testing is done while lock is held and when this macro exits the lock is held.

The lock is locked/unlocked using spin_lock/spin_unlock functions which must match the way they are locked/unlocked outside of this macro.

wake_up_locked has to be called after changing any variable that could change the result of the wait condition.

The function will return -ERESTARTSYS if it was interrupted by a signal and 0 if *condition* evaluated to true.

wait_event_interruptible_locked_irq

LINUX

Kernel Hackers ManualApril 2016

Name

wait_event_interruptible_locked_irq — sleep until a condition gets true

Synopsis

```
wait_event_interruptible_locked_irq ( wq,  condition);
```

Arguments

wq

the waitqueue to wait on

condition

a C expression for the event to wait for

Description

The process is put to sleep (TASK_INTERRUPTIBLE) until the *condition* evaluates to true or a signal is received. The *condition* is checked each time the waitqueue *wq* is woken up.

It must be called with wq.lock being held. This spinlock is unlocked while sleeping but *condition* testing is done while lock is held and when this macro exits the lock is held.

The lock is locked/unlocked using spin_lock_irq/spin_unlock_irq functions which must match the way they are locked/unlocked outside of this macro.

wake_up_locked has to be called after changing any variable that could change the result of the wait condition.

The function will return -ERESTARTSYS if it was interrupted by a signal and 0 if *condition* evaluated to true.

wait_event_interruptible_exclusive_locked

LINUX

Kernel Hackers ManualApril 2016

Name

wait_event_interruptible_exclusive_locked — sleep exclusively until a condition gets true

Synopsis

```
wait_event_interruptible_exclusive_locked ( wq,  condition);
```

Arguments

wq

the waitqueue to wait on

condition

a C expression for the event to wait for

Description

The process is put to sleep (TASK_INTERRUPTIBLE) until the *condition* evaluates to true or a signal is received. The *condition* is checked each time the waitqueue *wq* is woken up.

It must be called with wq.lock being held. This spinlock is unlocked while sleeping but *condition* testing is done while lock is held and when this macro exits the lock is held.

The lock is locked/unlocked using spin_lock/spin_unlock functions which must match the way they are locked/unlocked outside of this macro.

The process is put on the wait queue with an WQ_FLAG_EXCLUSIVE flag set thus when other process waits process on the list if this process is awaken further processes are not considered.

wake_up_locked has to be called after changing any variable that could change the result of the wait condition.

The function will return -ERESTARTSYS if it was interrupted by a signal and 0 if *condition* evaluated to true.

wait_event_interruptible_exclusive_locked_irq

LINUX

Kernel Hackers ManualApril 2016

Name

wait_event_interruptible_exclusive_locked_irq — sleep until a

condition gets true

Synopsis

```
wait_event_interruptible_exclusive_locked_irq ( wq,
 condition);
```

Arguments

wq

the waitqueue to wait on

condition

a C expression for the event to wait for

Description

The process is put to sleep (TASK_INTERRUPTIBLE) until the *condition* evaluates to true or a signal is received. The *condition* is checked each time the waitqueue *wq* is woken up.

It must be called with wq.lock being held. This spinlock is unlocked while sleeping but *condition* testing is done while lock is held and when this macro exits the lock is held.

The lock is locked/unlocked using spin_lock_irq/spin_unlock_irq functions which must match the way they are locked/unlocked outside of this macro.

The process is put on the wait queue with an WQ_FLAG_EXCLUSIVE flag set thus when other process waits process on the list if this process is awaken further processes are not considered.

wake_up_locked has to be called after changing any variable that could change the result of the wait condition.

The function will return -ERESTARTSYS if it was interrupted by a signal and 0 if *condition* evaluated to true.

wait_event_interruptible_lock_irq_timeout

LINUX

Kernel Hackers ManualApril 2016

Name

`wait_event_interruptible_lock_irq_timeout` — sleep until a condition gets true or a timeout elapses. The condition is checked under the lock. This is expected to be called with the lock taken.

Synopsis

```
wait_event_interruptible_lock_irq_timeout ( wq,  condition,
 lock,  timeout);
```

Arguments

wq

the waitqueue to wait on

condition

a C expression for the event to wait for

lock

a locked spinlock_t, which will be released before `schedule` and reacquired afterwards.

timeout

timeout, in jiffies

Description

The process is put to sleep (TASK_INTERRUPTIBLE) until the *condition* evaluates to true or signal is received. The *condition* is checked each time the waitqueue *wq* is woken up.

wake_up has to be called after changing any variable that could change the result of the wait condition.

This is supposed to be called while holding the lock. The lock is dropped before going to sleep and is reacquired afterwards.

The function returns 0 if the *timeout* elapsed, -ERESTARTSYS if it was interrupted by a signal, and the remaining jiffies otherwise if the condition evaluated to true before the timeout elapsed.

wait_event_killable

LINUX

Kernel Hackers ManualApril 2016

Name

wait_event_killable — sleep until a condition gets true

Synopsis

```
wait_event_killable ( wq,  condition);
```

Arguments

wq

the waitqueue to wait on

condition

a C expression for the event to wait for

Description

The process is put to sleep (TASK_KILLABLE) until the *condition* evaluates to true or a signal is received. The *condition* is checked each time the waitqueue *wq* is woken up.

wake_up has to be called after changing any variable that could change the result of the wait condition.

The function will return -ERESTARTSYS if it was interrupted by a signal and 0 if *condition* evaluated to true.

wait_on_bit

LINUX

Kernel Hackers ManualApril 2016

Name

wait_on_bit — wait for a bit to be cleared

Synopsis

```
int wait_on_bit (void * word, int bit, int (*action) (void *),
unsigned mode);
```

Arguments

word

the word being waited on, a kernel virtual address

bit

the bit of the word being waited on

action

the function used to sleep, which may take special actions

mode

the task state to sleep in

Description

There is a standard hashed waitqueue table for generic use. This is the part of the hashtable's accessor API that waits on a bit. For instance, if one were to have waiters on a bitflag, one would call `wait_on_bit` in threads waiting for the bit to clear. One uses `wait_on_bit` where one is waiting for the bit to clear, but has no intention of setting it.

wait_on_bit_lock

LINUX

Kernel Hackers ManualApril 2016

Name

`wait_on_bit_lock` — wait for a bit to be cleared, when wanting to set it

Synopsis

```
int wait_on_bit_lock (void * word, int bit, int (*action)
(void *), unsigned mode);
```

Arguments

word

the word being waited on, a kernel virtual address

bit

the bit of the word being waited on

action

the function used to sleep, which may take special actions

mode

the task state to sleep in

Description

There is a standard hashed waitqueue table for generic use. This is the part of the hashtable's accessor API that waits on a bit when one intends to set it, for instance, trying to lock bitflags. For instance, if one were to have waiters trying to set bitflag and waiting for it to clear before setting it, one would call `wait_on_bit` in threads waiting to be able to set the bit. One uses `wait_on_bit_lock` where one is waiting for the bit to clear with the intention of setting it, and when done, clearing it.

wait_on_atomic_t

LINUX

Kernel Hackers ManualApril 2016

Name

`wait_on_atomic_t` — Wait for an atomic_t to become 0

Synopsis

```
int wait_on_atomic_t (atomic_t * val, int (*action) (atomic_t
*), unsigned mode);
```

Arguments

val

The atomic value being waited on, a kernel virtual address

action

the function used to sleep, which may take special actions

mode

the task state to sleep in

Description

Wait for an atomic_t to become 0. We abuse the bit-wait waitqueue table for the purpose of getting a waitqueue, but we set the key to a bit number outside of the target 'word'.

finish_wait

LINUX

Kernel Hackers ManualApril 2016

Name

`finish_wait` — clean up after waiting in a queue

Synopsis

```
void finish_wait (wait_queue_head_t * q, wait_queue_t * wait);
```

Arguments

q

waitqueue waited on

wait

wait descriptor

Description

Sets current thread back to running state and removes the wait descriptor from the given waitqueue if still queued.

abort_exclusive_wait

LINUX

Kernel Hackers ManualApril 2016

Name

abort_exclusive_wait — abort exclusive waiting in a queue

Synopsis

```
void abort_exclusive_wait (wait_queue_head_t * q, wait_queue_t
* wait, unsigned int mode, void * key);
```

Arguments

q

waitqueue waited on

wait

wait descriptor

mode

runstate of the waiter to be woken

key

key to identify a wait bit queue or NULL

Description

Sets current thread back to running state and removes the wait descriptor from the given waitqueue if still queued.

Wakes up the next waiter if the caller is concurrently woken up through the queue.

This prevents waiter starvation where an exclusive waiter aborts and is woken up concurrently and no one wakes up the next waiter.

wake_up_bit

LINUX

Kernel Hackers ManualApril 2016

Name

wake_up_bit — wake up a waiter on a bit

Synopsis

```
void wake_up_bit (void * word, int bit);
```

Arguments

word

the word being waited on, a kernel virtual address

bit

the bit of the word being waited on

Description

There is a standard hashed waitqueue table for generic use. This is the part of the hashtable's accessor API that wakes up waiters on a bit. For instance, if one were to have waiters on a bitflag, one would call `wake_up_bit` after clearing the bit.

In order for this to function properly, as it uses `waitqueue_active` internally, some kind of memory barrier must be done prior to calling this. Typically, this will be `smp_mb__after_clear_bit`, but in some cases where bitflags are manipulated non-atomically under a lock, one may need to use a less regular barrier, such fs/inode.c's `smp_mb`, because `spin_unlock` does not guarantee a memory barrier.

wake_up_atomic_t

LINUX

Kernel Hackers ManualApril 2016

Name

`wake_up_atomic_t` — Wake up a waiter on a atomic_t

Synopsis

```
void wake_up_atomic_t (atomic_t * p);
```

Arguments

p

-- undescribed --

Description

Wake up anyone waiting for the atomic_t to go to zero.

Abuse the bit-waker function and its waitqueue hash table set (the atomic_t check is done by the waiter's wake function, not the by the waker itself).

1.5. High-resolution timers

ktime_set

LINUX

Kernel Hackers ManualApril 2016

Name

`ktime_set` — Set a ktime_t variable from a seconds/nanoseconds value

Synopsis

```
ktime_t ktime_set (const long secs, const unsigned long
nsecs);
```

Arguments

secs

seconds to set

nsecs

nanoseconds to set

Description

Return the ktime_t representation of the value

ktime_sub

LINUX

Kernel Hackers ManualApril 2016

Name

`ktime_sub` — subtract two ktime_t variables

Synopsis

```
ktime_t ktime_sub (const ktime_t lhs, const ktime_t rhs);
```

Arguments

lhs

minuend

rhs

subtrahend

Description

Returns the remainder of the subtraction

ktime_add

LINUX

Kernel Hackers ManualApril 2016

Name

ktime_add — add two ktime_t variables

Synopsis

```
ktime_t ktime_add (const ktime_t add1, const ktime_t add2);
```

Arguments

add1

addend1

add2

addend2

Description

Returns the sum of *add1* and *add2*.

timespec_to_ktime

LINUX

Kernel Hackers ManualApril 2016

Name

`timespec_to_ktime` — convert a timespec to ktime_t format

Synopsis

```
ktime_t timespec_to_ktime (const struct timespec ts);
```

Arguments

ts

the timespec variable to convert

Description

Returns a ktime_t variable with the converted timespec value

timeval_to_ktime

LINUX

Kernel Hackers ManualApril 2016

Name

`timeval_to_ktime` — convert a timeval to ktime_t format

Synopsis

```
ktime_t timeval_to_ktime (const struct timeval tv);
```

Arguments

tv

the timeval variable to convert

Description

Returns a ktime_t variable with the converted timeval value

ktime_to_timespec

LINUX

Kernel Hackers ManualApril 2016

Name

`ktime_to_timespec` — convert a ktime_t variable to timespec format

Synopsis

```
struct timespec ktime_to_timespec (const ktime_t kt);
```

Arguments

kt

the ktime_t variable to convert

Description

Returns the timespec representation of the ktime value

ktime_to_timeval

LINUX

Kernel Hackers ManualApril 2016

Name

`ktime_to_timeval` — convert a ktime_t variable to timeval format

Synopsis

```
struct timeval ktime_to_timeval (const ktime_t kt);
```

Arguments

kt

the ktime_t variable to convert

Description

Returns the timeval representation of the ktime value

ktime_to_ns

LINUX

Kernel Hackers ManualApril 2016

Name

`ktime_to_ns` — convert a ktime_t variable to scalar nanoseconds

Synopsis

```
s64 ktime_to_ns (const ktime_t kt);
```

Arguments

kt

the ktime_t variable to convert

Description

Returns the scalar nanoseconds representation of *kt*

ktime_equal

LINUX

Kernel Hackers ManualApril 2016

Name

`ktime_equal` — Compares two ktime_t variables to see if they are equal

Synopsis

```
int ktime_equal (const ktime_t cmp1, const ktime_t cmp2);
```

Arguments

cmp1

comparable1

cmp2

comparable2

Description

Compare two ktime_t variables, returns 1 if equal

struct hrtimer

LINUX

Kernel Hackers ManualApril 2016

Name

`struct hrtimer` — the basic hrtimer structure

Synopsis

```
struct hrtimer {
  struct timerqueue_node node;
  ktime_t _softexpires;
  enum hrtimer_restart  (* function) (struct hrtimer *);
  struct hrtimer_clock_base * base;
  unsigned long state;
#ifdef CONFIG_TIMER_STATS
  int start_pid;
  void * start_site;
  char start_comm[16];
#endif
};
```

Members

node

timerqueue node, which also manages node.expires, the absolute expiry time in the hrtimers internal representation. The time is related to the clock on which the timer is based. Is setup by adding slack to the _softexpires value. For non range timers identical to _softexpires.

_softexpires

the absolute earliest expiry time of the hrtimer. The time which was given as expiry time when the timer was armed.

function

timer expiry callback function

base

pointer to the timer base (per cpu and per clock)

state

state information (See bit values above)

start_pid

timer statistics field to store the pid of the task which started the timer

start_site

timer statistics field to store the site where the timer was started

start_comm[16]

timer statistics field to store the name of the process which started the timer

Description

The hrtimer structure must be initialized by `hrtimer_init`

struct hrtimer_sleeper

LINUX

Kernel Hackers ManualApril 2016

Name

`struct hrtimer_sleeper` — simple sleeper structure

Synopsis

```
struct hrtimer_sleeper {
  struct hrtimer timer;
  struct task_struct * task;
};
```

Members

timer

embedded timer structure

task

task to wake up

Description

task is set to NULL, when the timer expires.

struct hrtimer_clock_base

LINUX

Kernel Hackers ManualApril 2016

Name

`struct hrtimer_clock_base` — the timer base for a specific clock

Synopsis

```
struct hrtimer_clock_base {
  struct hrtimer_cpu_base * cpu_base;
  int index;
  clockid_t clockid;
  struct timerqueue_head active;
  ktime_t resolution;
  ktime_t (* get_time) (void);
  ktime_t softirq_time;
  ktime_t offset;
};
```

Members

cpu_base

per cpu clock base

index

clock type index for per_cpu support when moving a timer to a base on another cpu.

clockid

clock id for per_cpu support

active

red black tree root node for the active timers

resolution

the resolution of the clock, in nanoseconds

get_time

function to retrieve the current time of the clock

softirq_time

the time when running the hrtimer queue in the softirq

offset

offset of this clock to the monotonic base

ktime_add_ns

LINUX

Kernel Hackers ManualApril 2016

Name

`ktime_add_ns` — Add a scalar nanoseconds value to a ktime_t variable

Synopsis

```
ktime_t ktime_add_ns (const ktime_t kt, u64 nsec);
```

Arguments

kt

addend

nsec

the scalar nsec value to add

Description

Returns the sum of kt and nsec in ktime_t format

ktime_sub_ns

LINUX

Kernel Hackers ManualApril 2016

Name

ktime_sub_ns — Subtract a scalar nanoseconds value from a ktime_t variable

Synopsis

```
ktime_t ktime_sub_ns (const ktime_t kt, u64 nsec);
```

Arguments

kt

minuend

nsec

the scalar nsec value to subtract

Description

Returns the subtraction of *nsec* from *kt* in ktime_t format

hrtimer_forward

LINUX

Kernel Hackers ManualApril 2016

Name

hrtimer_forward — forward the timer expiry

Synopsis

```
u64 hrtimer_forward (struct hrtimer * timer, ktime_t now,
ktime_t interval);
```

Arguments

timer

hrtimer to forward

now

forward past this time

interval

the interval to forward

Description

Forward the timer expiry so it will expire in the future. Returns the number of overruns.

hrtimer_start_range_ns

LINUX

Kernel Hackers ManualApril 2016

Name

`hrtimer_start_range_ns` — (re)start an hrtimer on the current CPU

Synopsis

```
int hrtimer_start_range_ns (struct hrtimer * timer, ktime_t
tim, unsigned long delta_ns, const enum hrtimer_mode mode);
```

Arguments

timer

the timer to be added

tim

expiry time

delta_ns

"slack" range for the timer

mode

expiry mode: absolute (HRTIMER_ABS) or relative (HRTIMER_REL)

Returns

0 on success 1 when the timer was active

hrtimer_start

LINUX

Kernel Hackers ManualApril 2016

Name

`hrtimer_start` — (re)start an hrtimer on the current CPU

Synopsis

```
int hrtimer_start (struct hrtimer * timer, ktime_t tim, const
enum hrtimer_mode mode);
```

Arguments

timer

the timer to be added

tim

expiry time

mode

expiry mode: absolute (HRTIMER_ABS) or relative (HRTIMER_REL)

Returns

0 on success 1 when the timer was active

hrtimer_try_to_cancel

LINUX

Kernel Hackers ManualApril 2016

Name

`hrtimer_try_to_cancel` — try to deactivate a timer

Synopsis

```
int hrtimer_try_to_cancel (struct hrtimer * timer);
```

Arguments

timer

hrtimer to stop

Returns

0 when the timer was not active 1 when the timer was active -1 when the timer is currently excuting the callback function and cannot be stopped

hrtimer_cancel

LINUX

Kernel Hackers ManualApril 2016

Name

`hrtimer_cancel` — cancel a timer and wait for the handler to finish.

Synopsis

```
int hrtimer_cancel (struct hrtimer * timer);
```

Arguments

timer

the timer to be cancelled

Returns

0 when the timer was not active 1 when the timer was active

hrtimer_get_remaining

LINUX

Kernel Hackers ManualApril 2016

Name

`hrtimer_get_remaining` — get remaining time for the timer

Synopsis

```
ktime_t hrtimer_get_remaining (const struct hrtimer * timer);
```

Arguments

`timer`

the timer to read

hrtimer_init

LINUX

Kernel Hackers ManualApril 2016

Name

`hrtimer_init` — initialize a timer to the given clock

Synopsis

```
void hrtimer_init (struct hrtimer * timer, clockid_t clock_id,
enum hrtimer_mode mode);
```

Arguments

timer

the timer to be initialized

clock_id

the clock to be used

mode

timer mode abs/rel

hrtimer_get_res

LINUX

Kernel Hackers ManualApril 2016

Name

hrtimer_get_res — get the timer resolution for a clock

Synopsis

```
int hrtimer_get_res (const clockid_t which_clock, struct
timespec * tp);
```

Arguments

which_clock

which clock to query

tp

pointer to timespec variable to store the resolution

Description

Store the resolution of the clock selected by *which_clock* in the variable pointed to by *tp*.

schedule_hrtimeout_range

LINUX

Kernel Hackers ManualApril 2016

Name

schedule_hrtimeout_range — sleep until timeout

Synopsis

```
int __sched schedule_hrtimeout_range (ktime_t * expires,
unsigned long delta, const enum hrtimer_mode mode);
```

Arguments

expires

timeout value (ktime_t)

delta

slack in expires timeout (ktime_t)

mode

timer mode, HRTIMER_MODE_ABS or HRTIMER_MODE_REL

Description

Make the current task sleep until the given expiry time has elapsed. The routine will return immediately unless the current task state has been set (see `set_current_state`).

The *delta* argument gives the kernel the freedom to schedule the actual wakeup to a time that is both power and performance friendly. The kernel give the normal best effort behavior for "*expires*+*delta*", but may decide to fire the timer earlier, but no earlier than *expires*.

You can set the task state as follows -

`TASK_UNINTERRUPTIBLE` - at least *timeout* time is guaranteed to pass before the routine returns.

`TASK_INTERRUPTIBLE` - the routine may return early if a signal is delivered to the current task.

The current task state is guaranteed to be TASK_RUNNING when this routine returns.

Returns 0 when the timer has expired otherwise -EINTR

schedule_hrtimeout

LINUX

Kernel Hackers ManualApril 2016

Name

`schedule_hrtimeout` — sleep until timeout

Synopsis

```
int __sched schedule_hrtimeout (ktime_t * expires, const enum
hrtimer_mode mode);
```

Arguments

expires

timeout value (ktime_t)

mode

timer mode, HRTIMER_MODE_ABS or HRTIMER_MODE_REL

Description

Make the current task sleep until the given expiry time has elapsed. The routine will return immediately unless the current task state has been set (see `set_current_state`).

You can set the task state as follows -

`TASK_UNINTERRUPTIBLE` - at least *`timeout`* time is guaranteed to pass before the routine returns.

`TASK_INTERRUPTIBLE` - the routine may return early if a signal is delivered to the current task.

The current task state is guaranteed to be TASK_RUNNING when this routine returns.

Returns 0 when the timer has expired otherwise -EINTR

1.6. Workqueues and Kevents

queue_work

LINUX

Kernel Hackers ManualApril 2016

Name

`queue_work` — queue work on a workqueue

Synopsis

```
int queue_work (struct workqueue_struct * wq, struct
work_struct * work);
```

Arguments

wq

workqueue to use

work

work to queue

Description

Returns 0 if *work* was already on a queue, non-zero otherwise.

We queue the work to the CPU on which it was submitted, but if the CPU dies it can be processed by another CPU.

queue_work_on

LINUX

Kernel Hackers ManualApril 2016

Name

queue_work_on — queue work on specific cpu

Synopsis

```
int queue_work_on (int cpu, struct workqueue_struct * wq,
struct work_struct * work);
```

Arguments

cpu

CPU number to execute work on

wq

workqueue to use

work

work to queue

Description

Returns 0 if *work* was already on a queue, non-zero otherwise.

We queue the work to a specific CPU, the caller must ensure it can't go away.

queue_delayed_work

LINUX

Kernel Hackers ManualApril 2016

Name

`queue_delayed_work` — queue work on a workqueue after delay

Synopsis

```
int queue_delayed_work (struct workqueue_struct * wq, struct
delayed_work * dwork, unsigned long delay);
```

Arguments

wq

workqueue to use

dwork

delayable work to queue

delay

number of jiffies to wait before queueing

Description

Returns 0 if *work* was already on a queue, non-zero otherwise.

queue_delayed_work_on

LINUX

Kernel Hackers ManualApril 2016

Name

queue_delayed_work_on — queue work on specific CPU after delay

Synopsis

```
int queue_delayed_work_on (int cpu, struct workqueue_struct *
wq, struct delayed_work * dwork, unsigned long delay);
```

Arguments

cpu

CPU number to execute work on

wq

workqueue to use

dwork

work to queue

delay

number of jiffies to wait before queueing

Description

Returns 0 if *work* was already on a queue, non-zero otherwise.

flush_workqueue

LINUX

Kernel Hackers ManualApril 2016

Name

`flush_workqueue` — ensure that any scheduled work has run to completion.

Synopsis

```
void flush_workqueue (struct workqueue_struct * wq);
```

Arguments

wq

workqueue to flush

Description

Forces execution of the workqueue and blocks until its completion. This is typically used in driver shutdown handlers.

We sleep until all works which were queued on entry have been handled, but we are not livelocked by new incoming ones.

drain_workqueue

LINUX

Name

`drain_workqueue` — drain a workqueue

Synopsis

```
void drain_workqueue (struct workqueue_struct * wq);
```

Arguments

wq

workqueue to drain

Description

Wait until the workqueue becomes empty. While draining is in progress, only chain queueing is allowed. IOW, only currently pending or running work items on *wq* can queue further work items on it. *wq* is flushed repeatedly until it becomes empty. The number of flushing is detemined by the depth of chaining and should be relatively short. Whine if it takes too long.

flush_work

LINUX

Name

`flush_work` — wait for a work to finish executing the last queueing instance

Synopsis

```
bool flush_work (struct work_struct * work);
```

Arguments

work

the work to flush

Description

Wait until *work* has finished execution. This function considers only the last queueing instance of *work*. If *work* has been enqueued across different CPUs on a non-reentrant workqueue or on multiple workqueues, *work* might still be executing on return on some of the CPUs from earlier queueing.

If *work* was queued only on a non-reentrant, ordered or unbound workqueue, *work* is guaranteed to be idle on return if it hasn't been requeued since flush started.

RETURNS

`true` if `flush_work` waited for the work to finish execution, `false` if it was already idle.

flush_work_sync

LINUX

Kernel Hackers ManualApril 2016

Name

`flush_work_sync` — wait until a work has finished execution

Synopsis

```
bool flush_work_sync (struct work_struct * work);
```

Arguments

work

the work to flush

Description

Wait until *work* has finished execution. On return, it's guaranteed that all queueing instances of *work* which happened before this function is called are finished. In other words, if *work* hasn't been requeued since this function was called, *work* is guaranteed to be idle on return.

RETURNS

`true` if `flush_work_sync` waited for the work to finish execution, `false` if it was already idle.

cancel_work_sync

LINUX

Kernel Hackers ManualApril 2016

Name

`cancel_work_sync` — cancel a work and wait for it to finish

Synopsis

```
bool cancel_work_sync (struct work_struct * work);
```

Arguments

work

the work to cancel

Description

Cancel *work* and wait for its execution to finish. This function can be used even if the work re-queues itself or migrates to another workqueue. On return from this function, *work* is guaranteed to be not pending or executing on any CPU.

cancel_work_sync(delayed_work->work) must not be used for delayed_work's. Use `cancel_delayed_work_sync` instead.

The caller must ensure that the workqueue on which *work* was last queued can't be destroyed before this function returns.

RETURNS

`true` if *work* was pending, `false` otherwise.

flush_delayed_work

LINUX

Kernel Hackers ManualApril 2016

Name

`flush_delayed_work` — wait for a dwork to finish executing the last queueing

Synopsis

```
bool flush_delayed_work (struct delayed_work * dwork);
```

Arguments

dwork

the delayed work to flush

Description

Delayed timer is cancelled and the pending work is queued for immediate execution. Like `flush_work`, this function only considers the last queueing instance of *dwork*.

RETURNS

`true` if `flush_work` waited for the work to finish execution, `false` if it was already idle.

flush_delayed_work_sync

LINUX

Kernel Hackers ManualApril 2016

Name

`flush_delayed_work_sync` — wait for a dwork to finish

Synopsis

```
bool flush_delayed_work_sync (struct delayed_work * dwork);
```

Arguments

dwork

the delayed work to flush

Description

Delayed timer is cancelled and the pending work is queued for execution immediately. Other than timer handling, its behavior is identical to `flush_work_sync`.

RETURNS

`true` if `flush_work_sync` waited for the work to finish execution, `false` if it was already idle.

cancel_delayed_work_sync

LINUX

Kernel Hackers ManualApril 2016

Name

`cancel_delayed_work_sync` — cancel a delayed work and wait for it to finish

Synopsis

```
bool cancel_delayed_work_sync (struct delayed_work * dwork);
```

Arguments

dwork

the delayed work cancel

Description

This is `cancel_work_sync` for delayed works.

RETURNS

`true` if *dwork* was pending, `false` otherwise.

schedule_work

LINUX

Kernel Hackers ManualApril 2016

Name

`schedule_work` — put work task in global workqueue

Synopsis

```
int schedule_work (struct work_struct * work);
```

Arguments

work

job to be done

Description

Returns zero if *work* was already on the kernel-global workqueue and non-zero otherwise.

This puts a job in the kernel-global workqueue if it was not already queued and leaves it in the same position on the kernel-global workqueue otherwise.

schedule_delayed_work

LINUX

Kernel Hackers ManualApril 2016

Name

schedule_delayed_work — put work task in global workqueue after delay

Synopsis

```
int schedule_delayed_work (struct delayed_work * dwork,
unsigned long delay);
```

Arguments

dwork

job to be done

delay

number of jiffies to wait or 0 for immediate execution

Description

After waiting for a given time this puts a job in the kernel-global workqueue.

schedule_delayed_work_on

LINUX

Kernel Hackers ManualApril 2016

Name

`schedule_delayed_work_on` — queue work in global workqueue on CPU after delay

Synopsis

```
int schedule_delayed_work_on (int cpu, struct delayed_work * dwork, unsigned long delay);
```

Arguments

cpu

cpu to use

dwork

job to be done

delay

number of jiffies to wait

Description

After waiting for a given time this puts a job in the kernel-global workqueue on the specified CPU.

flush_scheduled_work

LINUX

Kernel Hackers ManualApril 2016

Name

`flush_scheduled_work` — ensure that any scheduled work has run to completion.

Synopsis

```
void flush_scheduled_work ( void);
```

Arguments

void

no arguments

Description

Forces execution of the kernel-global workqueue and blocks until its completion.

Think twice before calling this function! It's very easy to get into trouble if you don't take great care. Either of the following situations

will lead to deadlock

One of the work items currently on the workqueue needs to acquire a lock held by your code or its caller.

Your code is running in the context of a work routine.

They will be detected by lockdep when they occur, but the first might not occur very often. It depends on what work items are on the workqueue and what locks they need, which you have no control over.

In most situations flushing the entire workqueue is overkill; you merely need to know that a particular work item isn't queued and isn't running. In such cases you should use `cancel_delayed_work_sync` or `cancel_work_sync` instead.

execute_in_process_context

LINUX

Kernel Hackers ManualApril 2016

Name

`execute_in_process_context` — reliably execute the routine with user context

Synopsis

```
int execute_in_process_context (work_func_t fn, struct
execute_work * ew);
```

Arguments

fn

the function to execute

ew

guaranteed storage for the execute work structure (must be available when the work executes)

Description

Executes the function immediately if process context is available, otherwise schedules the function for delayed execution.

Returns

0 - function was executed 1 - function was scheduled for execution

destroy_workqueue

LINUX

Kernel Hackers ManualApril 2016

Name

`destroy_workqueue` — safely terminate a workqueue

Synopsis

```
void destroy_workqueue (struct workqueue_struct * wq);
```

Arguments

wq

target workqueue

Description

Safely destroy a workqueue. All work currently pending will be done first.

workqueue_set_max_active

LINUX

Kernel Hackers ManualApril 2016

Name

`workqueue_set_max_active` — adjust max_active of a workqueue

Synopsis

```
void workqueue_set_max_active (struct workqueue_struct * wq,
int max_active);
```

Arguments

wq

target workqueue

max_active

new max_active value.

Description

Set max_active of *wq* to *max_active*.

CONTEXT

Don't call from IRQ context.

workqueue_congested

LINUX

Kernel Hackers ManualApril 2016

Name

`workqueue_congested` — test whether a workqueue is congested

Synopsis

```
bool workqueue_congested (unsigned int cpu, struct
workqueue_struct * wq);
```

Arguments

cpu

CPU in question

wq

target workqueue

Description

Test whether *wq*'s cpu workqueue for *cpu* is congested. There is no synchronization around this function and the test result is unreliable and only useful as advisory hints or for debugging.

RETURNS

`true` if congested, `false` otherwise.

work_cpu

LINUX

Kernel Hackers ManualApril 2016

Name

`work_cpu` — return the last known associated cpu for *work*

Synopsis

```
unsigned int work_cpu (struct work_struct * work);
```

Arguments

work

the work of interest

RETURNS

CPU number if *work* was ever queued. WORK_CPU_NONE otherwise.

work_busy

LINUX

Kernel Hackers ManualApril 2016

Name

work_busy — test whether a work is currently pending or running

Synopsis

```
unsigned int work_busy (struct work_struct * work);
```

Arguments

work

the work to be tested

Description

Test whether *work* is currently pending or running. There is no synchronization around this function and the test result is unreliable and only useful as advisory hints or for debugging. Especially for reentrant wqs, the pending state might hide the running state.

RETURNS

OR'd bitmask of WORK_BUSY_* bits.

work_on_cpu

LINUX

Kernel Hackers ManualApril 2016

Name

work_on_cpu — run a function in user context on a particular cpu

Synopsis

```
long work_on_cpu (unsigned int cpu, long (*fn) (void *), void
* arg);
```

Arguments

cpu

the cpu to run on

fn

the function to run

arg

the function arg

Description

This will return the value *fn* returns. It is up to the caller to ensure that the cpu doesn't go offline. The caller must not hold any locks which would prevent *fn* from completing.

1.7. Internal Functions

reparent_to_kthreadd

LINUX

Kernel Hackers ManualApril 2016

Name

`reparent_to_kthreadd` — Reparent the calling kernel thread to kthreadd

Synopsis

```
void reparent_to_kthreadd ( void);
```

Arguments

void

no arguments

Description

If a kernel thread is launched as a result of a system call, or if it ever exits, it should generally reparent itself to kthreadd so it isn't in the way of other processes and is correctly cleaned up on exit.

The various task state such as scheduling policy and priority may have been inherited from a user process, so we reset them to sane values here.

NOTE that `reparent_to_kthreadd` gives the caller full capabilities.

wait_task_stopped

LINUX

Kernel Hackers ManualApril 2016

Name

`wait_task_stopped` — Wait for `TASK_STOPPED` or `TASK_TRACED`

Synopsis

```
int wait_task_stopped (struct wait_opts * wo, int ptrace,
struct task_struct * p);
```

Arguments

wo

wait options

ptrace

is the wait for ptrace

p

task to wait for

Description

Handle `sys_wait4` work for `p` in state `TASK_STOPPED` or `TASK_TRACED`.

CONTEXT

read_lock(tasklist_lock), which is released if return value is non-zero. Also, grabs and releases *p*->sighand->siglock.

RETURNS

0 if wait condition didn't exist and search for other wait conditions should continue. Non-zero return, -errno on failure and `p`'s pid on success, implies that tasklist_lock is released and wait condition search should terminate.

task_clear_group_stop_trapping

LINUX

Kernel Hackers ManualApril 2016

Name

`task_clear_group_stop_trapping` — clear group stop trapping bit

Synopsis

```
void task_clear_group_stop_trapping (struct task_struct * task);
```

Arguments

`task`

target task

Description

If GROUP_STOP_TRAPPING is set, a ptracer is waiting for us. Clear it and wake up the ptracer. Note that we don't need any further locking. *`task`*->siglock guarantees that *`task`*->parent points to the ptracer.

CONTEXT

Must be called with *task*->sighand->siglock held.

task_clear_group_stop_pending

LINUX

Kernel Hackers ManualApril 2016

Name

`task_clear_group_stop_pending` — clear pending group stop

Synopsis

```
void task_clear_group_stop_pending (struct task_struct *
task);
```

Arguments

task

target task

Description

Clear group stop states for *task*.

CONTEXT

Must be called with *task*->sighand->siglock held.

task_participate_group_stop

LINUX

Kernel Hackers ManualApril 2016

Name

`task_participate_group_stop` — participate in a group stop

Synopsis

```
bool task_participate_group_stop (struct task_struct * task);
```

Arguments

task

task participating in a group stop

Description

task has GROUP_STOP_PENDING set and is participating in a group stop. Group stop states are cleared and the group stop count is consumed if `GROUP_STOP_CONSUME` was set. If the consumption completes the group stop, the appropriate `SIGNAL_*` flags are set.

CONTEXT

Must be called with *task*->sighand->siglock held.

RETURNS

`true` if group stop completion should be notified to the parent, `false` otherwise.

do_notify_parent_cldstop

LINUX

Kernel Hackers ManualApril 2016

Name

`do_notify_parent_cldstop` — notify parent of stopped/continued state change

Synopsis

```
void do_notify_parent_cldstop (struct task_struct * tsk, bool for_ptracer, int why);
```

Arguments

`tsk`

task reporting the state change

`for_ptracer`

the notification is for ptracer

`why`

CLD_{CONTINUED|STOPPED|TRAPPED} to report

Description

Notify *tsk*'s parent that the stopped/continued state has changed. If *for_ptracer* is false, *tsk*'s group leader notifies to its real parent. If true, *tsk* reports to *tsk*->parent which should be the ptracer.

CONTEXT

Must be called with tasklist_lock at least read locked.

sys_restart_syscall

LINUX

Kernel Hackers ManualApril 2016

Name

sys_restart_syscall — restart a system call

Synopsis

```
long sys_restart_syscall ( void);
```

Arguments

void

no arguments

set_current_blocked

LINUX

Kernel Hackers ManualApril 2016

Name

`set_current_blocked` — change current->blocked mask

Synopsis

```
void set_current_blocked (const sigset_t * newset);
```

Arguments

`newset`

new mask

Description

It is wrong to change ->blocked directly, this helper should be used to ensure the process can't miss a shared signal we are going to block.

sys_rt_sigprocmask

LINUX

Name

`sys_rt_sigprocmask` — change the list of currently blocked signals

Synopsis

```
long sys_rt_sigprocmask (int how, sigset_t __user * nset,
sigset_t __user * oset, size_t sigsetsize);
```

Arguments

how

whether to add, remove, or set signals

nset

stores pending signals

oset

previous value of signal mask if non-null

sigsetsize

size of sigset_t type

sys_rt_sigpending

LINUX

Kernel Hackers ManualApril 2016

Name

`sys_rt_sigpending` — examine a pending signal that has been raised while blocked

Synopsis

```
long sys_rt_sigpending (sigset_t __user * set, size_t sigsetsize);
```

Arguments

set

stores pending signals

sigsetsize

size of sigset_t type or larger

do_sigtimedwait

LINUX

Kernel Hackers ManualApril 2016

Name

`do_sigtimedwait` — wait for queued signals specified in *which*

Synopsis

```
int do_sigtimedwait (const sigset_t * which, siginfo_t * info,
const struct timespec * ts);
```

Arguments

which

queued signals to wait for

info

if non-null, the signal's siginfo is returned here

ts

upper bound on process time suspension

sys_rt_sigtimedwait

LINUX

Kernel Hackers ManualApril 2016

Name

`sys_rt_sigtimedwait` — synchronously wait for queued signals specified in *uthese*

Synopsis

```
long sys_rt_sigtimedwait (const sigset_t __user * uthese,
siginfo_t __user * uinfo, const struct timespec __user * uts,
size_t sigsetsize);
```

Arguments

uthese

queued signals to wait for

uinfo

if non-null, the signal's siginfo is returned here

uts

upper bound on process time suspension

sigsetsize

size of sigset_t type

sys_kill

LINUX

Kernel Hackers ManualApril 2016

Name

`sys_kill` — send a signal to a process

Synopsis

```
long sys_kill (pid_t pid, int sig);
```

Arguments

pid

the PID of the process

sig

signal to be sent

sys_tgkill

LINUX

Kernel Hackers ManualApril 2016

Name

`sys_tgkill` — send signal to one specific thread

Synopsis

```
long sys_tgkill (pid_t tgid, pid_t pid, int sig);
```

Arguments

tgid

the thread group ID of the thread

pid

the PID of the thread

sig

signal to be sent

Description

This syscall also checks the *tgid* and returns -ESRCH even if the PID exists but it's not belonging to the target process anymore. This method solves the problem of threads exiting and PIDs getting reused.

sys_tkill

LINUX

Kernel Hackers ManualApril 2016

Name

sys_tkill — send signal to one specific task

Synopsis

```
long sys_tkill (pid_t pid, int sig);
```

Arguments

pid

the PID of the task

sig

signal to be sent

Description

Send a signal to only one task, even if it's a CLONE_THREAD task.

sys_rt_sigqueueinfo

LINUX

Kernel Hackers ManualApril 2016

Name

`sys_rt_sigqueueinfo` — send signal information to a signal

Synopsis

```
long sys_rt_sigqueueinfo (pid_t pid, int sig, siginfo_t __user
* uinfo);
```

Arguments

`pid`

the PID of the thread

`sig`

signal to be sent

`uinfo`

signal info to be sent

sys_sigpending

LINUX

Kernel Hackers ManualApril 2016

Name

`sys_sigpending` — examine pending signals

Synopsis

```
long sys_sigpending (old_sigset_t __user * set);
```

Arguments

set

where mask of pending signal is returned

sys_sigprocmask

LINUX

Kernel Hackers ManualApril 2016

Name

`sys_sigprocmask` — examine and change blocked signals

Synopsis

```
long sys_sigprocmask (int how, old_sigset_t __user * nset,
old_sigset_t __user * oset);
```

Arguments

how

whether to add, remove, or set signals

nset

signals to add or remove (if non-null)

oset

previous value of signal mask if non-null

Description

Some platforms have their own version with special arguments; others support only sys_rt_sigprocmask.

sys_rt_sigaction

LINUX

Kernel Hackers ManualApril 2016

Name

`sys_rt_sigaction` — alter an action taken by a process

Synopsis

```
long sys_rt_sigaction (int sig, const struct sigaction __user
* act, struct sigaction __user * oact, size_t sigsetsize);
```

Arguments

sig

signal to be sent

act

new sigaction

oact

used to save the previous sigaction

sigsetsize

size of sigset_t type

sys_rt_sigsuspend

LINUX

Kernel Hackers ManualApril 2016

Name

sys_rt_sigsuspend — replace the signal mask for a value with the *unewset* value until a signal is received

Synopsis

```
long sys_rt_sigsuspend (sigset_t __user * unewset, size_t
sigsetsize);
```

Arguments

unewset

new signal mask value

sigsetsize

size of sigset_t type

kthread_run

LINUX

Kernel Hackers ManualApril 2016

Name

kthread_run — create and wake a thread.

Synopsis

```
kthread_run ( threadfn,  data,  namefmt,  ...);
```

Arguments

threadfn

the function to run until signal_pending(current).

data

data ptr for *threadfn*.

namefmt

printf-style name for the thread.

`...`

variable arguments

Description

Convenient wrapper for `kthread_create` followed by `wake_up_process`. Returns the kthread or ERR_PTR(-ENOMEM).

kthread_should_stop

LINUX

Kernel Hackers ManualApril 2016

Name

`kthread_should_stop` — should this kthread return now?

Synopsis

```
int kthread_should_stop ( void);
```

Arguments

`void`

no arguments

Description

When someone calls `kthread_stop` on your kthread, it will be woken and this will return true. You should then return, and your return value will be passed through to `kthread_stop`.

kthread_create_on_node

LINUX

Kernel Hackers ManualApril 2016

Name

`kthread_create_on_node` — create a kthread.

Synopsis

```
struct task_struct * kthread_create_on_node (int (*threadfn)
(void *data), void * data, int node, const char namefmt[],
 ...);
```

Arguments

`threadfn`

the function to run until signal_pending(current).

`data`

data ptr for *`threadfn`*.

`node`

memory node number.

`namefmt[]`

printf-style name for the thread.

...

variable arguments

Description

This helper function creates and names a kernel thread. The thread will be stopped: use `wake_up_process` to start it. See also `kthread_run`.

If thread is going to be bound on a particular cpu, give its node in *`node`*, to get NUMA affinity for kthread stack, or else give -1. When woken, the thread will run *`threadfn`*() with *`data`* as its argument. *`threadfn`*() can either call `do_exit` directly if it is a standalone thread for which no one will call `kthread_stop`, or return when '`kthread_should_stop`' is true (which means `kthread_stop` has been called). The return value should be zero or a negative error number; it will be passed to `kthread_stop`.

Returns a task_struct or ERR_PTR(-ENOMEM).

kthread_bind

LINUX

Kernel Hackers ManualApril 2016

Name

`kthread_bind` — bind a just-created kthread to a cpu.

Synopsis

```
void kthread_bind (struct task_struct * p, unsigned int cpu);
```

Arguments

p

thread created by `kthread_create`.

cpu

cpu (might not be online, must be possible) for *k* to run on.

Description

This function is equivalent to `set_cpus_allowed`, except that *cpu* doesn't need to be online, and the thread must be stopped (i.e., just returned from `kthread_create`).

kthread_stop

LINUX

Kernel Hackers ManualApril 2016

Name

`kthread_stop` — stop a thread created by `kthread_create`.

Synopsis

```
int kthread_stop (struct task_struct * k);
```

Arguments

k

thread created by `kthread_create`.

Description

Sets `kthread_should_stop` for *k* to return true, wakes it, and waits for it to exit. This can also be called after `kthread_create` instead of calling `wake_up_process`: the thread will exit without calling `threadfn`.

If `threadfn` may call `do_exit` itself, the caller must ensure task_struct can't go away.

Returns the result of `threadfn`, or `-EINTR` if `wake_up_process` was never called.

kthread_worker_fn

LINUX

Kernel Hackers ManualApril 2016

Name

`kthread_worker_fn` — kthread function to process kthread_worker

Synopsis

```
int kthread_worker_fn (void * worker_ptr);
```

Arguments

worker_ptr

pointer to initialized kthread_worker

Description

This function can be used as *threadfn* to kthread_create or kthread_run with *worker_ptr* argument pointing to an initialized kthread_worker. The started kthread will process work_list until the it is stopped with kthread_stop. A kthread can also call this function directly after extra initialization.

Different kthreads can be used for the same kthread_worker as long as there's only one kthread attached to it at any given time. A kthread_worker without an attached kthread simply collects queued kthread_works.

queue_kthread_work

LINUX

Kernel Hackers ManualApril 2016

Name

queue_kthread_work — queue a kthread_work

Synopsis

```
bool queue_kthread_work (struct kthread_worker * worker,
struct kthread_work * work);
```

Arguments

worker

target kthread_worker

work

kthread_work to queue

Description

Queue *work* to work processor *task* for async execution. *task* must have been created with `kthread_worker_create`. Returns `true` if *work* was successfully queued, `false` if it was already pending.

flush_kthread_work

LINUX

Kernel Hackers ManualApril 2016

Name

`flush_kthread_work` — flush a kthread_work

Synopsis

```
void flush_kthread_work (struct kthread_work * work);
```

Arguments

work

work to flush

Description

If *work* is queued or executing, wait for it to finish execution.

flush_kthread_worker

LINUX

Kernel Hackers ManualApril 2016

Name

`flush_kthread_worker` — flush all current works on a kthread_worker

Synopsis

```
void flush_kthread_worker (struct kthread_worker * worker);
```

Arguments

worker

worker to flush

Description

Wait until all currently executing or pending works on *worker* are finished.

1.8. Kernel objects manipulation

kobject_get_path

LINUX

Name

`kobject_get_path` — generate and return the path associated with a given kobj and kset pair.

Synopsis

```
char * kobject_get_path (struct kobject * kobj, gfp_t
gfp_mask);
```

Arguments

kobj

kobject in question, with which to build the path

gfp_mask

the allocation type used to allocate the path

Description

The result must be freed by the caller with `kfree`.

kobject_set_name

LINUX

Name

`kobject_set_name` — Set the name of a kobject

Synopsis

```
int kobject_set_name (struct kobject * kobj, const char * fmt,
 ...);
```

Arguments

`kobj`

struct kobject to set the name of

`fmt`

format string used to build the name

`...`

variable arguments

Description

This sets the name of the kobject. If you have already added the kobject to the system, you must call `kobject_rename` in order to change the name of the kobject.

kobject_init

LINUX

Name

`kobject_init` — initialize a kobject structure

Synopsis

```
void kobject_init (struct kobject * kobj, struct kobj_type *
ktype);
```

Arguments

`kobj`

pointer to the kobject to initialize

`ktype`

pointer to the ktype for this kobject.

Description

This function will properly initialize a kobject such that it can then be passed to the `kobject_add` call.

After this function is called, the kobject MUST be cleaned up by a call to `kobject_put`, not by a call to kfree directly to ensure that all of the memory is cleaned up properly.

kobject_add

LINUX

Name

`kobject_add` — the main kobject add function

Synopsis

```
int kobject_add (struct kobject * kobj, struct kobject *
parent, const char * fmt,  ...);
```

Arguments

`kobj`

the kobject to add

`parent`

pointer to the parent of the kobject.

`fmt`

format to name the kobject with.

`...`

variable arguments

Description

The kobject name is set and added to the kobject hierarchy in this function.

If *`parent`* is set, then the parent of the *`kobj`* will be set to it. If *`parent`* is NULL, then the parent of the *`kobj`* will be set to the kobject associted with the kset assigned to this kobject. If no kset is assigned to the kobject, then the kobject will be located in the root of the sysfs tree.

If this function returns an error, `kobject_put` must be called to properly clean up the memory associated with the object. Under no instance should the kobject that is passed to this function be directly freed with a call to `kfree`, that can leak memory.

Note, no “add” uevent will be created with this call, the caller should set up all of the necessary sysfs files for the object and then call `kobject_uevent` with the UEVENT_ADD parameter to ensure that userspace is properly notified of this kobject’s creation.

kobject_init_and_add

LINUX

Kernel Hackers ManualApril 2016

Name

`kobject_init_and_add` — initialize a kobject structure and add it to the kobject hierarchy

Synopsis

```
int kobject_init_and_add (struct kobject * kobj, struct
kobj_type * ktype, struct kobject * parent, const char * fmt,
 ...);
```

Arguments

`kobj`

pointer to the kobject to initialize

`ktype`

pointer to the ktype for this kobject.

`parent`

pointer to the parent of this kobject.

fmt

the name of the kobject.

...

variable arguments

Description

This function combines the call to `kobject_init` and `kobject_add`. The same type of error handling after a call to `kobject_add` and kobject lifetime rules are the same here.

kobject_rename

LINUX

Kernel Hackers ManualApril 2016

Name

`kobject_rename` — change the name of an object

Synopsis

```
int kobject_rename (struct kobject * kobj, const char *
new_name);
```

Arguments

kobj

object in question.

new_name

object's new name

Description

It is the responsibility of the caller to provide mutual exclusion between two different calls of kobject_rename on the same kobject and to ensure that new_name is valid and won't conflict with other kobjects.

kobject_del

LINUX

Kernel Hackers ManualApril 2016

Name

`kobject_del` — unlink kobject from hierarchy.

Synopsis

```
void kobject_del (struct kobject * kobj);
```

Arguments

kobj

object.

kobject_get

LINUX

Kernel Hackers ManualApril 2016

Name

`kobject_get` — increment refcount for object.

Synopsis

```
struct kobject * kobject_get (struct kobject * kobj);
```

Arguments

`kobj`

object.

kobject_put

LINUX

Kernel Hackers ManualApril 2016

Name

`kobject_put` — decrement refcount for object.

Synopsis

```
void kobject_put (struct kobject * kobj);
```

Arguments

kobj

object.

Description

Decrement the refcount, and if 0, call `kobject_cleanup`.

kobject_create_and_add

LINUX

Kernel Hackers ManualApril 2016

Name

`kobject_create_and_add` — create a struct kobject dynamically and register it with sysfs

Synopsis

```
struct kobject * kobject_create_and_add (const char * name,
struct kobject * parent);
```

Arguments

name

the name for the kset

parent

the parent kobject of this kobject, if any.

Description

This function creates a kobject structure dynamically and registers it with sysfs. When you are finished with this structure, call `kobject_put` and the structure will be dynamically freed when it is no longer being used.

If the kobject was not able to be created, NULL will be returned.

kset_register

LINUX

Kernel Hackers ManualApril 2016

Name

`kset_register` — initialize and add a kset.

Synopsis

```
int kset_register (struct kset * k);
```

Arguments

k

kset.

kset_unregister

LINUX

Kernel Hackers ManualApril 2016

Name

`kset_unregister` — remove a kset.

Synopsis

```
void kset_unregister (struct kset * k);
```

Arguments

k

kset.

kset_create_and_add

LINUX

Name

`kset_create_and_add` — create a struct kset dynamically and add it to sysfs

Synopsis

```
struct kset * kset_create_and_add (const char * name, const
struct kset_uevent_ops * uevent_ops, struct kobject *
parent_kobj);
```

Arguments

`name`

the name for the kset

`uevent_ops`

a struct kset_uevent_ops for the kset

`parent_kobj`

the parent kobject of this kset, if any.

Description

This function creates a kset structure dynamically and registers it with sysfs. When you are finished with this structure, call `kset_unregister` and the structure will be dynamically freed when it is no longer being used.

If the kset was not able to be created, NULL will be returned.

1.9. Kernel utility functions

upper_32_bits

LINUX

Kernel Hackers ManualApril 2016

Name

`upper_32_bits` — return bits 32-63 of a number

Synopsis

```
upper_32_bits ( n);
```

Arguments

n

the number we're accessing

Description

A basic shift-right of a 64- or 32-bit quantity. Use this to suppress the "right shift count >= width of type" warning when that quantity is 32-bits.

lower_32_bits

LINUX

Name

`lower_32_bits` — return bits 0-31 of a number

Synopsis

```
lower_32_bits ( n);
```

Arguments

n

the number we're accessing

might_sleep

LINUX

Name

`might_sleep` — annotation for functions that can sleep

Synopsis

```
might_sleep (void);
```

Arguments

None

Description

this macro will print a stack trace if it is executed in an atomic context (spinlock, irq-handler, ...).

This is a useful debugging help to be able to catch problems early and not be bitten later when the calling function happens to sleep when it is not supposed to.

trace_printk

LINUX

Kernel Hackers ManualApril 2016

Name

`trace_printk` — printf formatting in the ftrace buffer

Synopsis

```
trace_printk ( fmt,  args...);
```

Arguments

`fmt`

the printf format for printing

`args...`

variable arguments

Note

__trace_printk is an internal function for trace_printk and the `ip` is passed in via the trace_printk macro.

This function allows a kernel developer to debug fast path sections that printk is not appropriate for. By scattering in various printk like tracing in the code, a developer can quickly see where problems are occurring.

This is intended as a debugging tool for the developer only. Please refrain from leaving trace_printks scattered around in your code.

min_not_zero

LINUX

Kernel Hackers ManualApril 2016

Name

`min_not_zero` — return the minimum that is _not_ zero, unless both are zero

Synopsis

```
min_not_zero ( x,  y);
```

Arguments

x

value1

y

value2

clamp

LINUX

Kernel Hackers ManualApril 2016

Name

clamp — return a value clamped to a given range with strict typechecking

Synopsis

```
clamp ( val,  min,  max);
```

Arguments

val

current value

min

minimum allowable value

max

maximum allowable value

Description

This macro does strict typechecking of min/max to make sure they are of the same type as val. See the unnecessary pointer comparisons.

clamp_t

LINUX

Kernel Hackers ManualApril 2016

Name

clamp_t — return a value clamped to a given range using a given type

Synopsis

```
clamp_t ( type,  val,  min,  max);
```

Arguments

type

the type of variable to use

val

current value

min

minimum allowable value

max

maximum allowable value

Description

This macro does no typechecking and uses temporary variables of type 'type' to make all the comparisons.

clamp_val

LINUX

Kernel Hackers ManualApril 2016

Name

`clamp_val` — return a value clamped to a given range using val's type

Synopsis

```
clamp_val ( val,  min,  max);
```

Arguments

val

current value

min

minimum allowable value

max

maximum allowable value

Description

This macro does no typechecking and uses temporary variables of whatever type the input argument 'val' is. This is useful when val is an unsigned type and min and max are literals that will otherwise be assigned a signed integer type.

container_of

LINUX

Kernel Hackers ManualApril 2016

Name

`container_of` — cast a member of a structure out to the containing structure

Synopsis

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

Arguments

`ptr`

the pointer to the member.

`type`

the type of the container struct this is embedded in.

`member`

the name of the member within the struct.

BUILD_BUG_ON

LINUX

Name

BUILD_BUG_ON — break compile if a condition is true.

Synopsis

```
BUILD_BUG_ON ( condition);
```

Arguments

condition

the condition which the compiler should know is false.

Description

If you have some code which relies on certain constants being equal, or other compile-time-evaluated condition, you should use BUILD_BUG_ON to detect if someone changes it.

The implementation uses gcc's reluctance to create a negative array, but gcc (as of 4.4) only emits that error for obvious cases (eg. not arguments to inline functions). So as a fallback we use the optimizer; if it can't prove the condition is false, it will cause a link error on the undefined "__build_bug_on_failed". This error message can be harder to track down though, hence the two different methods.

printk

LINUX

Name

`printk` — print a kernel message

Synopsis

```
int printk (const char * fmt,  ...);
```

Arguments

`fmt`

format string

`...`

variable arguments

Description

This is `printk`. It can be called from any context. We want it to work.

We try to grab the console_lock. If we succeed, it's easy - we log the output and call the console drivers. If we fail to get the semaphore we place the output into the log buffer and return. The current holder of the console_sem will notice the new output in `console_unlock`; and will send it to the consoles before releasing the lock.

One effect of this deferred printing is that code which calls `printk` and then changes console_loglevel may break. This is because console_loglevel is inspected when the actual printing occurs.

See also

printf(3)

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

console_lock

LINUX

Kernel Hackers ManualApril 2016

Name

`console_lock` — lock the console system for exclusive use.

Synopsis

```
void console_lock ( void);
```

Arguments

void

no arguments

Description

Acquires a lock which guarantees that the caller has exclusive access to the console system and the console_drivers list.

Can sleep, returns nothing.

console_trylock

LINUX

Name

`console_trylock` — try to lock the console system for exclusive use.

Synopsis

```
int console_trylock ( void);
```

Arguments

void

no arguments

Description

Tried to acquire a lock which guarantees that the caller has exclusive access to the console system and the console_drivers list.

returns 1 on success, and 0 on failure to acquire the lock.

console_conditional_schedule

LINUX

Name

`console_conditional_schedule` — yield the CPU if required

Synopsis

```
void __sched console_conditional_schedule ( void);
```

Arguments

void

no arguments

Description

If the console code is currently allowed to sleep, and if this CPU should yield the CPU to another task, do so here.

Must be called within `console_lock`;.

printk_timed_ratelimit

LINUX

Kernel Hackers ManualApril 2016

Name

`printk_timed_ratelimit` — caller-controlled printk ratelimiting

Synopsis

```
bool printk_timed_ratelimit (unsigned long * caller_jiffies,
unsigned int interval_msecs);
```

Arguments

caller_jiffies

pointer to caller's state

interval_msecs

minimum interval between prints

Description

printk_timed_ratelimit returns true if more than *interval_msecs* milliseconds have elapsed since the last time printk_timed_ratelimit returned true.

kmsg_dump_register

LINUX

Kernel Hackers ManualApril 2016

Name

kmsg_dump_register — register a kernel log dumper.

Synopsis

```
int kmsg_dump_register (struct kmsg_dumper * dumper);
```

Arguments

dumper

pointer to the kmsg_dumper structure

Description

Adds a kernel log dumper to the system. The dump callback in the structure will be called when the kernel oopses or panics and must be set. Returns zero on success and `-EINVAL` or `-EBUSY` otherwise.

kmsg_dump_unregister

LINUX

Kernel Hackers ManualApril 2016

Name

`kmsg_dump_unregister` — unregister a kmsg dumper.

Synopsis

```
int kmsg_dump_unregister (struct kmsg_dumper * dumper);
```

Arguments

`dumper`

pointer to the kmsg_dumper structure

Description

Removes a dump device from the system. Returns zero on success and `-EINVAL` otherwise.

panic

LINUX

Kernel Hackers ManualApril 2016

Name

`panic` — halt the system

Synopsis

```
NORET_TYPE void panic (const char * fmt,  ...);
```

Arguments

`fmt`

The text string to print

`...`

variable arguments

Description

Display a message, then perform cleanups.

This function never returns.

emergency_restart

LINUX

Name

`emergency_restart` — reboot the system

Synopsis

```
void emergency_restart ( void);
```

Arguments

void

no arguments

Description

Without shutting down any hardware or taking any locks reboot the system. This is called when we know we are in trouble so this is our best effort to reboot. This is safe to call in interrupt context.

kernel_restart

LINUX

Name

`kernel_restart` — reboot the system

Synopsis

```
void kernel_restart (char * cmd);
```

Arguments

cmd

pointer to buffer containing command to execute for restart or NULL

Description

Shutdown everything and perform a clean reboot. This is not safe to call in interrupt context.

kernel_halt

LINUX

Kernel Hackers ManualApril 2016

Name

kernel_halt — halt the system

Synopsis

```
void kernel_halt ( void);
```

Arguments

void

no arguments

Description

Shutdown everything and perform a clean system halt.

kernel_power_off

LINUX

Kernel Hackers ManualApril 2016

Name

`kernel_power_off` — power_off the system

Synopsis

```
void kernel_power_off ( void);
```

Arguments

void

no arguments

Description

Shutdown everything and perform a clean system power_off.

orderly_poweroff

LINUX

Kernel Hackers ManualApril 2016

Name

`orderly_poweroff` — Trigger an orderly system poweroff

Synopsis

```
int orderly_poweroff (bool force);
```

Arguments

force

force poweroff if command execution fails

Description

This may be called from any context to trigger a system shutdown. If the orderly shutdown fails, it will force an immediate shutdown.

rcu_read_lock_bh_held

LINUX

Kernel Hackers ManualApril 2016

Name

`rcu_read_lock_bh_held` — might we be in RCU-bh read-side critical section?

Synopsis

```
int rcu_read_lock_bh_held ( void);
```

Arguments

void

no arguments

Description

Check for bottom half being disabled, which covers both the CONFIG_PROVE_RCU and not cases. Note that if someone uses `rcu_read_lock_bh`, but then later enables BH, lockdep (if enabled) will show the situation. This is useful for debug checks in functions that require that they be called within an RCU read-side critical section.

Check `debug_lockdep_rcu_enabled` to prevent false positives during boot.

init_rcu_head_on_stack

LINUX

Kernel Hackers ManualApril 2016

Name

`init_rcu_head_on_stack` — initialize on-stack rcu_head for debugobjects

Synopsis

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

Arguments

`head`

pointer to rcu_head structure to be initialized

Description

This function informs debugobjects of a new rcu_head structure that has been allocated as an auto variable on the stack. This function is not required for rcu_head structures that are statically defined or that are dynamically allocated on the heap. This function has no effect for !CONFIG_DEBUG_OBJECTS_RCU_HEAD kernel builds.

destroy_rcu_head_on_stack

LINUX

Name

`destroy_rcu_head_on_stack` — destroy on-stack rcu_head for debugobjects

Synopsis

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

Arguments

head

pointer to rcu_head structure to be initialized

Description

This function informs debugobjects that an on-stack rcu_head structure is about to go out of scope. As with `init_rcu_head_on_stack`, this function is not required for rcu_head structures that are statically defined or that are dynamically allocated on the heap. Also as with `init_rcu_head_on_stack`, this function has no effect for !CONFIG_DEBUG_OBJECTS_RCU_HEAD kernel builds.

1.10. Device Resource Management

devres_alloc

LINUX

Name

devres_alloc — Allocate device resource data

Synopsis

```
void * devres_alloc (dr_release_t release, size_t size, gfp_t
gfp);
```

Arguments

release

Release function devres will be associated with

size

Allocation size

gfp

Allocation flags

Description

Allocate devres of *size* bytes. The allocated area is zeroed, then associated with *release*. The returned pointer can be passed to other devres_*() functions.

RETURNS

Pointer to allocated devres on success, NULL on failure.

devres_for_each_res

LINUX

Kernel Hackers ManualApril 2016

Name

devres_for_each_res — Resource iterator

Synopsis

```
void devres_for_each_res (struct device * dev, dr_release_t
release, dr_match_t match, void * match_data, void (*fn)
(struct device *, void *, void *), void * data);
```

Arguments

dev

Device to iterate resource from

release

Look for resources associated with this release function

match

Match function (optional)

match_data

Data for the match function

fn

Function to be called for each matched resource.

data

Data for *fn*, the 3rd parameter of *fn*

Description

Call *fn* for each devres of *dev* which is associated with *release* and for which *match* returns 1.

RETURNS

void

devres_free

LINUX

Kernel Hackers ManualApril 2016

Name

devres_free — Free device resource data

Synopsis

```
void devres_free (void * res);
```

Arguments

res

Pointer to devres data to free

Description

Free devres created with devres_alloc.

devres_add

LINUX

Kernel Hackers ManualApril 2016

Name

devres_add — Register device resource

Synopsis

```
void devres_add (struct device * dev, void * res);
```

Arguments

dev

Device to add resource to

res

Resource to register

Description

Register devres *res* to *dev*. *res* should have been allocated using devres_alloc. On driver detach, the associated release function will be invoked and devres will be freed automatically.

devres_find

LINUX

Kernel Hackers ManualApril 2016

Name

devres_find — Find device resource

Synopsis

```
void * devres_find (struct device * dev, dr_release_t release,
dr_match_t match, void * match_data);
```

Arguments

dev

Device to lookup resource from

release

Look for resources associated with this release function

match

Match function (optional)

match_data

Data for the match function

Description

Find the latest devres of *dev* which is associated with *release* and for which *match* returns 1. If *match* is NULL, it's considered to match all.

RETURNS

Pointer to found devres, NULL if not found.

devres_get

LINUX

Kernel Hackers ManualApril 2016

Name

`devres_get` — Find devres, if non-existent, add one atomically

Synopsis

```
void * devres_get (struct device * dev, void * new_res,
dr_match_t match, void * match_data);
```

Arguments

dev

Device to lookup or add devres for

new_res

Pointer to new initialized devres to add if not found

match

Match function (optional)

match_data

Data for the match function

Description

Find the latest devres of *dev* which has the same release function as *new_res* and for which *match* return 1. If found, *new_res* is freed; otherwise, *new_res* is added atomically.

RETURNS

Pointer to found or added devres.

devres_remove

LINUX

Kernel Hackers ManualApril 2016

Name

devres_remove — Find a device resource and remove it

Synopsis

```
void * devres_remove (struct device * dev, dr_release_t
release, dr_match_t match, void * match_data);
```

Arguments

dev

Device to find resource from

release

Look for resources associated with this release function

match

Match function (optional)

match_data

Data for the match function

Description

Find the latest devres of *dev* associated with *release* and for which *match* returns 1. If *match* is NULL, it's considered to match all. If found, the resource is removed atomically and returned.

RETURNS

Pointer to removed devres on success, NULL if not found.

devres_destroy

LINUX

Kernel Hackers ManualApril 2016

Name

devres_destroy — Find a device resource and destroy it

Synopsis

```
int devres_destroy (struct device * dev, dr_release_t release,
dr_match_t match, void * match_data);
```

Arguments

dev

Device to find resource from

release

Look for resources associated with this release function

match

Match function (optional)

match_data

Data for the match function

Description

Find the latest devres of *dev* associated with *release* and for which *match* returns 1. If *match* is NULL, it's considered to match all. If found, the resource is removed atomically and freed.

RETURNS

0 if devres is found and freed, -ENOENT if not found.

devres_open_group

LINUX

Kernel Hackers ManualApril 2016

Name

devres_open_group — Open a new devres group

Synopsis

```
void * devres_open_group (struct device * dev, void * id,
gfp_t gfp);
```

Arguments

dev

Device to open devres group for

id

Separator ID

gfp

Allocation flags

Description

Open a new devres group for *dev* with *id*. For *id*, using a pointer to an object which won't be used for another group is recommended. If *id* is NULL, address-wise unique ID is created.

RETURNS

ID of the new group, NULL on failure.

devres_close_group

LINUX

Name

`devres_close_group` — Close a devres group

Synopsis

```
void devres_close_group (struct device * dev, void * id);
```

Arguments

`dev`

Device to close devres group for

`id`

ID of target group, can be NULL

Description

Close the group identified by *`id`*. If *`id`* is NULL, the latest open group is selected.

devres_remove_group

LINUX

Name

`devres_remove_group` — Remove a devres group

Synopsis

```
void devres_remove_group (struct device * dev, void * id);
```

Arguments

dev

Device to remove group for

id

ID of target group, can be NULL

Description

Remove the group identified by *id*. If *id* is NULL, the latest open group is selected. Note that removing a group doesn't affect any other resources.

devres_release_group

LINUX

Kernel Hackers ManualApril 2016

Name

devres_release_group — Release resources in a devres group

Synopsis

```
int devres_release_group (struct device * dev, void * id);
```

Arguments

dev

Device to release group for

id

ID of target group, can be NULL

Description

Release all resources in the group identified by *id*. If *id* is NULL, the latest open group is selected. The selected group and groups properly nested inside the selected group are removed.

RETURNS

The number of released non-group resources.

devm_kzalloc

LINUX

Kernel Hackers ManualApril 2016

Name

devm_kzalloc — Resource-managed kzalloc

Synopsis

```
void * devm_kzalloc (struct device * dev, size_t size, gfp_t
gfp);
```

Arguments

dev

Device to allocate memory for

size

Allocation size

gfp

Allocation gfp flags

Description

Managed kzalloc. Memory allocated with this function is automatically freed on driver detach. Like all other devres resources, guaranteed alignment is unsigned long long.

RETURNS

Pointer to allocated memory on success, NULL on failure.

devm_kfree

LINUX

Kernel Hackers ManualApril 2016

Name

`devm_kfree` — Resource-managed kfree

Synopsis

```
void devm_kfree (struct device * dev, void * p);
```

Arguments

dev

Device this memory belongs to

p

Memory to free

Description

Free memory allocated with `dev_kzalloc`.

Chapter 2. Device drivers infrastructure

2.1. The Basic Device Driver-Model Structures

struct bus_type

LINUX

Kernel Hackers ManualApril 2016

Name

`struct bus_type` — The bus type of the device

Synopsis

```
struct bus_type {
  const char * name;
  struct bus_attribute * bus_attrs;
  struct device_attribute * dev_attrs;
  struct driver_attribute * drv_attrs;
  int (* match) (struct device *dev, struct device_driver *drv);
  int (* uevent) (struct device *dev, struct kobj_uevent_env *env);
  int (* probe) (struct device *dev);
  int (* remove) (struct device *dev);
  void (* shutdown) (struct device *dev);
  int (* suspend) (struct device *dev, pm_message_t state);
  int (* resume) (struct device *dev);
  const struct dev_pm_ops * pm;
  struct subsys_private * p;
};
```

Members

name

The name of the bus.

bus_attrs

Default attributes of the bus.

dev_attrs

Default attributes of the devices on the bus.

drv_attrs

Default attributes of the device drivers on the bus.

match

Called, perhaps multiple times, whenever a new device or driver is added for this bus. It should return a nonzero value if the given device can be handled by the given driver.

uevent

Called when a device is added, removed, or a few other things that generate uevents to add the environment variables.

probe

Called when a new device or driver add to this bus, and callback the specific driver’s probe to initial the matched device.

remove

Called when a device removed from this bus.

shutdown

Called at shut-down time to quiesce the device.

suspend

Called when a device on this bus wants to go to sleep mode.

resume

Called to bring a device on this bus out of sleep mode.

pm

Power management operations of this bus, callback the specific device driver's pm-ops.

p

The private data of the driver core, only the driver core can touch this.

Description

A bus is a channel between the processor and one or more devices. For the purposes of the device model, all devices are connected via a bus, even if it is an internal, virtual, "platform" bus. Buses can plug into each other. A USB controller is usually a PCI device, for example. The device model represents the actual connections between buses and the devices they control. A bus is represented by the bus_type structure. It contains the name, the default attributes, the bus' methods, PM operations, and the driver core's private data.

struct device_driver

LINUX

Kernel Hackers ManualApril 2016

Name

`struct device_driver` — The basic device driver structure

Synopsis

```
struct device_driver {
  const char * name;
  struct bus_type * bus;
  struct module * owner;
  const char * mod_name;
  bool suppress_bind_attrs;
  const struct of_device_id * of_match_table;
  int (* probe) (struct device *dev);
  int (* remove) (struct device *dev);
```

```
  void (* shutdown) (struct device *dev);
  int (* suspend) (struct device *dev, pm_message_t state);
  int (* resume) (struct device *dev);
  const struct attribute_group ** groups;
  const struct dev_pm_ops * pm;
  struct driver_private * p;
};
```

Members

name

Name of the device driver.

bus

The bus which the device of this driver belongs to.

owner

The module owner.

mod_name

Used for built-in modules.

suppress_bind_attrs

Disables bind/unbind via sysfs.

of_match_table

The open firmware table.

probe

Called to query the existence of a specific device, whether this driver can work with it, and bind the driver to a specific device.

remove

Called when the device is removed from the system to unbind a device from this driver.

shutdown

Called at shut-down time to quiesce the device.

suspend

Called to put the device to sleep mode. Usually to a low power state.

resume

Called to bring a device from sleep mode.

groups

Default attributes that get created by the driver core automatically.

pm

Power management operations of the device which matched this driver.

p

Driver core's private data, no one other than the driver core can touch this.

Description

The device driver-model tracks all of the drivers known to the system. The main reason for this tracking is to enable the driver core to match up drivers with new devices. Once drivers are known objects within the system, however, a number of other things become possible. Device drivers can export information and configuration variables that are independent of any specific device.

struct class

LINUX

Kernel Hackers ManualApril 2016

Name

`struct class` — device classes

Synopsis

```
struct class {
```

```
  const char * name;
  struct module * owner;
  struct class_attribute * class_attrs;
  struct device_attribute * dev_attrs;
  struct bin_attribute * dev_bin_attrs;
  struct kobject * dev_kobj;
  int (* dev_uevent) (struct device *dev, struct kobj_uevent_env *env);
  char *(* devnode) (struct device *dev, mode_t *mode);
  void (* class_release) (struct class *class);
  void (* dev_release) (struct device *dev);
  int (* suspend) (struct device *dev, pm_message_t state);
  int (* resume) (struct device *dev);
  const struct kobj_ns_type_operations * ns_type;
  const void *(* namespace) (struct device *dev);
  const struct dev_pm_ops * pm;
  struct subsys_private * p;
};
```

Members

name

Name of the class.

owner

The module owner.

class_attrs

Default attributes of this class.

dev_attrs

Default attributes of the devices belong to the class.

dev_bin_attrs

Default binary attributes of the devices belong to the class.

dev_kobj

The kobject that represents this class and links it into the hierarchy.

dev_uevent

Called when a device is added, removed from this class, or a few other things that generate uevents to add the environment variables.

devnode

Callback to provide the devtmpfs.

class_release

Called to release this class.

dev_release

Called to release the device.

suspend

Used to put the device to sleep mode, usually to a low power state.

resume

Used to bring the device from the sleep mode.

ns_type

Callbacks so sysfs can detemine namespaces.

namespace

Namespace of the device belongs to this class.

pm

The default device power management operations of this class.

p

The private data of the driver core, no one other than the driver core can touch this.

Description

A class is a higher-level view of a device that abstracts out low-level implementation details. Drivers may see a SCSI disk or an ATA disk, but, at the class level, they are all simply disks. Classes allow user space to work with devices based on what they do, rather than how they are connected or how they work.

struct device

LINUX

Kernel Hackers ManualApril 2016

Name

`struct device` — The basic device structure

Synopsis

```
struct device {
  struct device * parent;
  struct device_private * p;
  struct kobject kobj;
  const char * init_name;
  const struct device_type * type;
  struct mutex mutex;
  struct bus_type * bus;
  struct device_driver * driver;
  void * platform_data;
  struct dev_pm_info power;
  struct dev_power_domain * pwr_domain;
#ifdef CONFIG_NUMA
  int numa_node;
#endif
  u64 * dma_mask;
  u64 coherent_dma_mask;
  struct device_dma_parameters * dma_parms;
  struct list_head dma_pools;
  struct dma_coherent_mem * dma_mem;
  struct dev_archdata archdata;
  struct device_node * of_node;
  dev_t devt;
  spinlock_t devres_lock;
  struct list_head devres_head;
  struct klist_node knode_class;
  struct class * class;
  const struct attribute_group ** groups;
  void (* release) (struct device *dev);
};
```

Members

parent

The device's "parent" device, the device to which it is attached. In most cases, a parent device is some sort of bus or host controller. If parent is NULL, the device, is a top-level device, which is not usually what you want.

p

Holds the private data of the driver core portions of the device. See the comment of the struct device_private for detail.

kobj

A top-level, abstract class from which other classes are derived.

init_name

Initial name of the device.

type

The type of device. This identifies the device type and carries type-specific information.

mutex

Mutex to synchronize calls to its driver.

bus

Type of bus device is on.

driver

Which driver has allocated this

platform_data

Platform data specific to the device.

power

For device power management. See Documentation/power/devices.txt for details.

pwr_domain

Provide callbacks that are executed during system suspend, hibernation, system resume and during runtime PM transitions along with subsystem-level and driver-level callbacks.

numa_node

NUMA node this device is close to.

dma_mask

Dma mask (if dma'ble device).

coherent_dma_mask

Like dma_mask, but for alloc_coherent mapping as not all hardware supports 64-bit addresses for consistent allocations such descriptors.

dma_parms

A low level driver may set these to teach IOMMU code about segment limitations.

dma_pools

Dma pools (if dma'ble device).

dma_mem

Internal for coherent mem override.

archdata

For arch-specific additions.

of_node

Associated device tree node.

devt

For creating the sysfs "dev".

devres_lock

Spinlock to protect the resource of the device.

devres_head

The resources list of the device.

knode_class

The node used to add the device to the class list.

class

The class of the device.

groups

Optional attribute groups.

release

Callback to free the device after all references have gone away. This should be set by the allocator of the device (i.e. the bus driver that discovered the device).

Example

```
For devices on custom boards, as typical of embedded
and SOC based hardware, Linux often uses platform_data to point
to board-specific structures describing devices and how they
are wired.  That can include what ports are available, chip
variants, which GPIO pins act in what additional roles, and so
on.  This shrinks the "Board Support Packages" (BSPs) and
minimizes board-specific #ifdefs in drivers.
```

Description

At the lowest level, every device in a Linux system is represented by an instance of struct device. The device structure contains the information that the device model core needs to model the system. Most subsystems, however, track additional information about the devices they host. As a result, it is rare for devices to be represented by bare device structures; instead, that structure, like kobject structures, is usually embedded within a higher-level representation of the device.

module_driver

LINUX

Kernel Hackers ManualApril 2016

Name

`module_driver` — Helper macro for drivers that don't do anything special in module init/exit. This eliminates a lot of boilerplate. Each module may only use this

macro once, and calling it replaces `module_init` and `module_exit`.

Synopsis

```
module_driver ( __driver,  __register,  __unregister,  ...);
```

Arguments

__driver

driver name

__register

register function for this driver type

__unregister

unregister function for this driver type @...: Additional arguments to be passed to __register and __unregister.

...

variable arguments

Description

Use this macro to construct bus specific macros for registering drivers, and do not use it on its own.

2.2. Device Drivers Base

driver_for_each_device

LINUX

Kernel Hackers ManualApril 2016

Name

`driver_for_each_device` — Iterator for devices bound to a driver.

Synopsis

```
int driver_for_each_device (struct device_driver * drv, struct
device * start, void * data, int (*fn) (struct device *, void
*));
```

Arguments

drv

Driver we're iterating.

start

Device to begin with

data

Data to pass to the callback.

fn

Function to call for each device.

Description

Iterate over the *drv*'s list of devices calling *fn* for each one.

driver_find_device

LINUX

Kernel Hackers ManualApril 2016

Name

driver_find_device — device iterator for locating a particular device.

Synopsis

```
struct device * driver_find_device (struct device_driver *
drv, struct device * start, void * data, int (*match) (struct
device *dev, void *data));
```

Arguments

drv

The device's driver

start

Device to begin with

data

Data to pass to match function

match

Callback function to check device

Description

This is similar to the `driver_for_each_device` function above, but it returns a reference to a device that is 'found' for later use, as determined by the *`match`* callback.

The callback should return 0 if the device doesn't match and non-zero if it does. If the callback returns non-zero, this function will return to the caller and not iterate over any more devices.

driver_create_file

LINUX

Kernel Hackers ManualApril 2016

Name

`driver_create_file` — create sysfs file for driver.

Synopsis

```
int driver_create_file (struct device_driver * drv, const
struct driver_attribute * attr);
```

Arguments

`drv`

driver.

`attr`

driver attribute descriptor.

driver_remove_file

LINUX

Kernel Hackers ManualApril 2016

Name

`driver_remove_file` — remove sysfs file for driver.

Synopsis

```
void driver_remove_file (struct device_driver * drv, const
struct driver_attribute * attr);
```

Arguments

drv

driver.

attr

driver attribute descriptor.

driver_add_kobj

LINUX

Kernel Hackers ManualApril 2016

Name

`driver_add_kobj` — add a kobject below the specified driver

Synopsis

```
int driver_add_kobj (struct device_driver * drv, struct
kobject * kobj, const char * fmt,  ...);
```

Arguments

drv

requesting device driver

kobj

kobject to add below this driver

fmt

format string that names the kobject

...

variable arguments

Description

You really don't want to do this, this is only here due to one looney iseries driver, go poke those developers if you are annoyed about this...

get_driver

LINUX

Kernel Hackers ManualApril 2016

Name

`get_driver` — increment driver reference count.

Synopsis

```
struct device_driver * get_driver (struct device_driver *
drv);
```

Arguments

drv

driver.

put_driver

LINUX

Kernel Hackers ManualApril 2016

Name

put_driver — decrement driver's refcount.

Synopsis

```
void put_driver (struct device_driver * drv);
```

Arguments

drv

driver.

driver_register

LINUX

Kernel Hackers ManualApril 2016

Name

`driver_register` — register driver with bus

Synopsis

```
int driver_register (struct device_driver * drv);
```

Arguments

drv

driver to register

Description

We pass off most of the work to the `bus_add_driver` call, since most of the things we have to do deal with the bus structures.

driver_unregister

LINUX

Name

`driver_unregister` — remove driver from system.

Synopsis

```
void driver_unregister (struct device_driver * drv);
```

Arguments

drv

driver.

Description

Again, we pass off most of the work to the bus-level call.

driver_find

LINUX

Name

`driver_find` — locate driver on a bus by its name.

Synopsis

```
struct device_driver * driver_find (const char * name, struct
bus_type * bus);
```

Arguments

name

name of the driver.

bus

bus to scan for the driver.

Description

Call `kset_find_obj` to iterate over list of drivers on a bus to find driver by name. Return driver if found.

Note that kset_find_obj increments driver's reference count.

dev_driver_string

LINUX

Kernel Hackers ManualApril 2016

Name

`dev_driver_string` — Return a device's driver name, if at all possible

Synopsis

```
const char * dev_driver_string (const struct device * dev);
```

Arguments

dev

struct device to get the name of

Description

Will return the device's driver's name if it is bound to a device. If the device is not bound to a device, it will return the name of the bus it is attached to. If it is not attached to a bus either, an empty string will be returned.

device_create_file

LINUX

Kernel Hackers ManualApril 2016

Name

`device_create_file` — create sysfs attribute file for device.

Synopsis

```
int device_create_file (struct device * dev, const struct
device_attribute * attr);
```

Arguments

dev

device.

attr

device attribute descriptor.

device_remove_file

LINUX

Kernel Hackers ManualApril 2016

Name

`device_remove_file` — remove sysfs attribute file.

Synopsis

```
void device_remove_file (struct device * dev, const struct
device_attribute * attr);
```

Arguments

dev

device.

attr

device attribute descriptor.

device_create_bin_file

LINUX

Kernel Hackers ManualApril 2016

Name

`device_create_bin_file` — create sysfs binary attribute file for device.

Synopsis

```
int device_create_bin_file (struct device * dev, const struct
bin_attribute * attr);
```

Arguments

dev

device.

attr

device binary attribute descriptor.

device_remove_bin_file

LINUX

Kernel Hackers ManualApril 2016

Name

`device_remove_bin_file` — remove sysfs binary attribute file

Synopsis

```
void device_remove_bin_file (struct device * dev, const struct
bin_attribute * attr);
```

Arguments

dev

device.

attr

device binary attribute descriptor.

device_schedule_callback_owner

LINUX

Kernel Hackers ManualApril 2016

Name

`device_schedule_callback_owner` — helper to schedule a callback for a device

Synopsis

```
int device_schedule_callback_owner (struct device * dev, void
(*func) (struct device *), struct module * owner);
```

Arguments

dev

device.

func

callback function to invoke later.

owner

module owning the callback routine

Description

Attribute methods must not unregister themselves or their parent device (which would amount to the same thing). Attempts to do so will deadlock, since unregistration is mutually exclusive with driver callbacks.

Instead methods can call this routine, which will attempt to allocate and schedule a workqueue request to call back *func* with *dev* as its argument in the workqueue's process context. *dev* will be pinned until *func* returns.

This routine is usually called via the inline device_schedule_callback, which automatically sets *owner* to THIS_MODULE.

Returns 0 if the request was submitted, -ENOMEM if storage could not be allocated, -ENODEV if a reference to *owner* isn't available.

NOTE

This routine won't work if CONFIG_SYSFS isn't set! It uses an underlying sysfs routine (since it is intended for use by attribute methods), and if sysfs isn't available you'll get nothing but -ENOSYS.

device_initialize

LINUX

Name

`device_initialize` — init device structure.

Synopsis

```
void device_initialize (struct device * dev);
```

Arguments

dev

device.

Description

This prepares the device for use by other layers by initializing its fields. It is the first half of `device_register`, if called by that function, though it can also be called separately, so one may use *dev*'s fields. In particular, `get_device`/`put_device` may be used for reference counting of *dev* after calling this function.

NOTE

Use `put_device` to give up your reference instead of freeing *dev* directly once you have called this function.

dev_set_name

LINUX

Name

`dev_set_name` — set a device name

Synopsis

```
int dev_set_name (struct device * dev, const char * fmt,
 ...);
```

Arguments

`dev`

device

`fmt`

format string for the device's name

`...`

variable arguments

device_add

LINUX

Kernel Hackers ManualApril 2016

Name

`device_add` — add device to device hierarchy.

Synopsis

```
int device_add (struct device * dev);
```

Arguments

dev

device.

Description

This is part 2 of `device_register`, though may be called separately _iff_ `device_initialize` has been called separately.

This adds *dev* to the kobject hierarchy via `kobject_add`, adds it to the global and sibling lists for the device, then adds it to the other relevant subsystems of the driver model.

NOTE

Never directly free *dev* after calling this function, even if it returned an error! Always use `put_device` to give up your reference instead.

device_register

LINUX

Kernel Hackers ManualApril 2016

Name

`device_register` — register a device with the system.

Synopsis

```
int device_register (struct device * dev);
```

Arguments

dev

pointer to the device structure

Description

This happens in two clean steps - initialize the device and add it to the system. The two steps can be called separately, but this is the easiest and most common. I.e. you should only call the two helpers separately if have a clearly defined need to use and refcount the device before it is added to the hierarchy.

NOTE

Never directly free *dev* after calling this function, even if it returned an error! Always use `put_device` to give up the reference initialized in this function instead.

get_device

LINUX

Kernel Hackers ManualApril 2016

Name

`get_device` — increment reference count for device.

Synopsis

```
struct device * get_device (struct device * dev);
```

Arguments

dev

device.

Description

This simply forwards the call to `kobject_get`, though we do take care to provide for the case that we get a NULL pointer passed in.

put_device

LINUX

Kernel Hackers ManualApril 2016

Name

`put_device` — decrement reference count.

Synopsis

```
void put_device (struct device * dev);
```

Arguments

dev

device in question.

device_del

LINUX

Kernel Hackers ManualApril 2016

Name

device_del — delete device from system.

Synopsis

```
void device_del (struct device * dev);
```

Arguments

dev

device.

Description

This is the first part of the device unregistration sequence. This removes the device from the lists we control from here, has it removed from the other driver model subsystems it was added to in device_add, and removes it from the kobject hierarchy.

NOTE

this should be called manually _iff_ `device_add` was also called manually.

device_unregister

LINUX

Kernel Hackers ManualApril 2016

Name

`device_unregister` — unregister device from system.

Synopsis

```
void device_unregister (struct device * dev);
```

Arguments

dev

device going away.

Description

We do this in two parts, like we do `device_register`. First, we remove it from all the subsystems with `device_del`, then we decrement the reference count via `put_device`. If that is the final reference count, the device will be cleaned up via `device_release` above. Otherwise, the structure will stick around until the final reference to the device is dropped.

device_for_each_child

LINUX

Kernel Hackers ManualApril 2016

Name

`device_for_each_child` — device child iterator.

Synopsis

```
int device_for_each_child (struct device * parent, void *
data, int (*fn) (struct device *dev, void *data));
```

Arguments

parent

parent struct device.

data

data for the callback.

fn

function to be called for each device.

Description

Iterate over *parent*'s child devices, and call *fn* for each, passing it *data*.

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

device_find_child

LINUX

Kernel Hackers ManualApril 2016

Name

`device_find_child` — device iterator for locating a particular device.

Synopsis

```
struct device * device_find_child (struct device * parent,
void * data, int (*match) (struct device *dev, void *data));
```

Arguments

parent

parent struct device

data

Data to pass to match function

match

Callback function to check device

Description

This is similar to the `device_for_each_child` function above, but it returns a reference to a device that is 'found' for later use, as determined by the *match* callback.

The callback should return 0 if the device doesn't match and non-zero if it does. If the callback returns non-zero and a reference to the current device can be obtained, this function will return to the caller and not iterate over any more devices.

__root_device_register

LINUX

Kernel Hackers ManualApril 2016

Name

__root_device_register — allocate and register a root device

Synopsis

```
struct device * __root_device_register (const char * name,
struct module * owner);
```

Arguments

name

root device name

owner

owner module of the root device, usually THIS_MODULE

Description

This function allocates a root device and registers it using device_register. In order to free the returned device, use root_device_unregister.

Root devices are dummy devices which allow other devices to be grouped under /sys/devices. Use this function to allocate a root device and then use it as the parent of any device which should appear under /sys/devices/{name}

The /sys/devices/{name} directory will also contain a 'module' symlink which points to the *owner* directory in sysfs.

Returns struct device pointer on success, or ERR_PTR on error.

Note

You probably want to use root_device_register.

root_device_unregister

LINUX

Kernel Hackers ManualApril 2016

Name

root_device_unregister — unregister and free a root device

Synopsis

```
void root_device_unregister (struct device * dev);
```

Arguments

dev

device going away

Description

This function unregisters and cleans up a device that was created by root_device_register.

device_create_vargs

LINUX

Kernel Hackers ManualApril 2016

Name

`device_create_vargs` — creates a device and registers it with sysfs

Synopsis

```
struct device * device_create_vargs (struct class * class,
struct device * parent, dev_t devt, void * drvdata, const char
* fmt, va_list args);
```

Arguments

class

pointer to the struct class that this device should be registered to

parent

pointer to the parent struct device of this new device, if any

devt

the dev_t for the char device to be added

drvdata

the data to be added to the device for callbacks

fmt

string for the device's name

args

va_list for the device's name

Description

This function can be used by char device classes. A struct device will be created in sysfs, registered to the specified class.

A “dev” file will be created, showing the dev_t for the device, if the dev_t is not 0,0. If a pointer to a parent struct device is passed in, the newly created struct device will be a child of that device in sysfs. The pointer to the struct device will be returned from the call. Any further sysfs files that might be required can be created using this pointer.

Returns struct device pointer on success, or `ERR_PTR` on error.

Note

the struct class passed to this function must have previously been created with a call to `class_create`.

device_create

LINUX

Kernel Hackers ManualApril 2016

Name

`device_create` — creates a device and registers it with sysfs

Synopsis

```
struct device * device_create (struct class * class, struct
device * parent, dev_t devt, void * drvdata, const char * fmt,
 ...);
```

Arguments

class

pointer to the struct class that this device should be registered to

parent

pointer to the parent struct device of this new device, if any

devt

the dev_t for the char device to be added

drvdata

the data to be added to the device for callbacks

fmt

string for the device's name

...

variable arguments

Description

This function can be used by char device classes. A struct device will be created in sysfs, registered to the specified class.

A "dev" file will be created, showing the dev_t for the device, if the dev_t is not 0,0. If a pointer to a parent struct device is passed in, the newly created struct device will be a child of that device in sysfs. The pointer to the struct device will be returned from the call. Any further sysfs files that might be required can be created using this pointer.

Returns struct device pointer on success, or `ERR_PTR` on error.

Note

the struct class passed to this function must have previously been created with a call to `class_create`.

device_destroy

LINUX

Kernel Hackers ManualApril 2016

Name

`device_destroy` — removes a device that was created with `device_create`

Synopsis

```
void device_destroy (struct class * class, dev_t devt);
```

Arguments

`class`

pointer to the struct class that this device was registered with

`devt`

the dev_t of the device that was previously registered

Description

This call unregisters and cleans up a device that was created with a call to `device_create`.

device_rename

LINUX

Name

`device_rename` — renames a device

Synopsis

```
int device_rename (struct device * dev, const char *
new_name);
```

Arguments

dev

the pointer to the struct device to be renamed

new_name

the new name of the device

Description

It is the responsibility of the caller to provide mutual exclusion between two different calls of device_rename on the same device to ensure that new_name is valid and won't conflict with other devices.

Note

Don't call this function. Currently, the networking layer calls this function, but that will change. The following text from Kay Sievers offers

some insight

Renaming devices is racy at many levels, symlinks and other stuff are not replaced atomically, and you get a "move" uevent, but it's not easy to connect the event to the

old and new device. Device nodes are not renamed at all, there isn't even support for that in the kernel now.

In the meantime, during renaming, your target name might be taken by another driver, creating conflicts. Or the old name is taken directly after you renamed it -- then you get events for the same DEVPATH, before you even see the "move" event. It's just a mess, and nothing new should ever rely on kernel device renaming. Besides that, it's not even implemented now for other things than (driver-core wise very simple) network devices.

We are currently about to change network renaming in udev to completely disallow renaming of devices in the same namespace as the kernel uses, because we can't solve the problems properly, that arise with swapping names of multiple interfaces without races. Means, renaming of eth[0-9]* will only be allowed to some other name than eth[0-9]*, for the aforementioned reasons.

Make up a "real" name in the driver before you register anything, or add some other attributes for userspace to find the device, or use udev to add symlinks -- but never rename kernel devices later, it's a complete mess. We don't even want to get into that and try to implement the missing pieces in the core. We really have other pieces to fix in the driver core mess. :)

device_move

LINUX

Kernel Hackers ManualApril 2016

Name

`device_move` — moves a device to a new parent

Synopsis

```
int device_move (struct device * dev, struct device *
new_parent, enum dpm_order dpm_order);
```

Arguments

dev

the pointer to the struct device to be moved

new_parent

the new parent of the device (can by NULL)

dpm_order

how to reorder the dpm_list

__class_create

LINUX

Kernel Hackers ManualApril 2016

Name

__class_create — create a struct class structure

Synopsis

```
struct class * __class_create (struct module * owner, const
char * name, struct lock_class_key * key);
```

Arguments

owner

pointer to the module that is to “own” this struct class

name

pointer to a string for the name of this class.

key

the lock_class_key for this class; used by mutex lock debugging

Description

This is used to create a struct class pointer that can then be used in calls to `device_create`.

Returns struct class pointer on success, or `ERR_PTR` on error.

Note, the pointer created here is to be destroyed when finished by making a call to `class_destroy`.

class_destroy

LINUX

Kernel Hackers ManualApril 2016

Name

`class_destroy` — destroys a struct class structure

Synopsis

```
void class_destroy (struct class * cls);
```

Arguments

cls

pointer to the struct class that is to be destroyed

Description

Note, the pointer to be destroyed must have been created with a call to `class_create`.

class_dev_iter_init

LINUX

Kernel Hackers ManualApril 2016

Name

`class_dev_iter_init` — initialize class device iterator

Synopsis

```
void class_dev_iter_init (struct class_dev_iter * iter, struct
class * class, struct device * start, const struct device_type
* type);
```

Arguments

iter

class iterator to initialize

class

the class we wanna iterate over

start

the device to start iterating from, if any

type

device_type of the devices to iterate over, NULL for all

Description

Initialize class iterator *iter* such that it iterates over devices of *class*. If *start* is set, the list iteration will start there, otherwise if it is NULL, the iteration starts at the beginning of the list.

class_dev_iter_next

LINUX

Kernel Hackers ManualApril 2016

Name

class_dev_iter_next — iterate to the next device

Synopsis

```
struct device * class_dev_iter_next (struct class_dev_iter *
iter);
```

Arguments

iter

class iterator to proceed

Description

Proceed *iter* to the next device and return it. Returns NULL if iteration is complete.

The returned device is referenced and won't be released till iterator is proceed to the next device or exited. The caller is free to do whatever it wants to do with the device including calling back into class code.

class_dev_iter_exit

LINUX

Kernel Hackers ManualApril 2016

Name

`class_dev_iter_exit` — finish iteration

Synopsis

```
void class_dev_iter_exit (struct class_dev_iter * iter);
```

Arguments

iter

class iterator to finish

Description

Finish an iteration. Always call this function after iteration is complete whether the iteration ran till the end or not.

class_for_each_device

LINUX

Kernel Hackers ManualApril 2016

Name

`class_for_each_device` — device iterator

Synopsis

```
int class_for_each_device (struct class * class, struct device
* start, void * data, int (*fn) (struct device *, void *));
```

Arguments

class

the class we're iterating

start

the device to start with in the list, if any.

data

data for the callback

fn

function to be called for each device

Description

Iterate over *class*'s list of devices, and call *fn* for each, passing it *data*. If *start* is set, the list iteration will start there, otherwise if it is NULL, the iteration starts at the beginning of the list.

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

fn is allowed to do anything including calling back into class code. There's no locking restriction.

class_find_device

LINUX

Kernel Hackers ManualApril 2016

Name

class_find_device — device iterator for locating a particular device

Synopsis

```
struct device * class_find_device (struct class * class,
struct device * start, void * data, int (*match) (struct
device *, void *));
```

Arguments

class

the class we're iterating

start

Device to begin with

data

data for the match function

match

function to check device

Description

This is similar to the `class_for_each_dev` function above, but it returns a reference to a device that is 'found' for later use, as determined by the *match* callback.

The callback should return 0 if the device doesn't match and non-zero if it does. If the callback returns non-zero, this function will return to the caller and not iterate over any more devices.

Note, you will need to drop the reference with `put_device` after use.

fn is allowed to do anything including calling back into class code. There's no locking restriction.

class_compat_register

LINUX

Kernel Hackers ManualApril 2016

Name

`class_compat_register` — register a compatibility class

Synopsis

```
struct class_compat * class_compat_register (const char *
name);
```

Arguments

name

the name of the class

Description

Compatibility class are meant as a temporary user-space compatibility workaround when converting a family of class devices to a bus devices.

class_compat_unregister

LINUX

Kernel Hackers ManualApril 2016

Name

`class_compat_unregister` — unregister a compatibility class

Synopsis

```
void class_compat_unregister (struct class_compat * cls);
```

Arguments

cls

the class to unregister

class_compat_create_link

LINUX

Kernel Hackers ManualApril 2016

Name

`class_compat_create_link` — create a compatibility class device link to a bus device

Synopsis

```
int class_compat_create_link (struct class_compat * cls,
struct device * dev, struct device * device_link);
```

Arguments

`cls`

the compatibility class

`dev`

the target bus device

`device_link`

an optional device to which a “device” link should be created

class_compat_remove_link

LINUX

Name

`class_compat_remove_link` — remove a compatibility class device link to a bus device

Synopsis

```
void class_compat_remove_link (struct class_compat * cls,
struct device * dev, struct device * device_link);
```

Arguments

`cls`

the compatibility class

`dev`

the target bus device

`device_link`

an optional device to which a “device” link was previously created

request_firmware

LINUX

Name

`request_firmware` — send firmware request and wait for it

Synopsis

```
int request_firmware (const struct firmware ** firmware_p,
const char * name, struct device * device);
```

Arguments

firmware_p

pointer to firmware image

name

name of firmware file

device

device for which firmware is being loaded

Description

firmware_p will be used to return a firmware image by the name of *name* for device *device*.

Should be called from user context where sleeping is allowed.

name will be used as $FIRMWARE in the uevent environment and should be distinctive enough not to be confused with any other firmware image for this or any other device.

Caller must hold the reference count of *device*.

release_firmware

LINUX

Name

`release_firmware` — release the resource associated with a firmware image

Synopsis

```
void release_firmware (const struct firmware * fw);
```

Arguments

fw

firmware resource to release

request_firmware_nowait

LINUX

Name

`request_firmware_nowait` — asynchronous version of request_firmware

Synopsis

```
int request_firmware_nowait (struct module * module, bool
uevent, const char * name, struct device * device, gfp_t gfp,
void * context, void (*cont) (const struct firmware *fw, void
*context));
```

Arguments

module

module requesting the firmware

uevent

sends uevent to copy the firmware image if this flag is non-zero else the firmware copy must be done manually.

name

name of firmware file

device

device for which firmware is being loaded

gfp

allocation flags

context

will be passed over to *cont*, and *fw* may be NULL if firmware request fails.

cont

function will be called asynchronously when the firmware request is over.

Description

Caller must hold the reference count of *device*.

Asynchronous variant of request_firmware for user contexts: - sleep for as small periods as possible since it may increase kernel boot time of built-in device drivers requesting firmware in their ->probe methods, if *gfp* is GFP_KERNEL.

- can't sleep at all if *gfp* is GFP_ATOMIC.

cache_firmware

LINUX

Kernel Hackers ManualApril 2016

Name

`cache_firmware` — cache one firmware image in kernel memory space

Synopsis

```
int cache_firmware (const char * fw_name);
```

Arguments

`fw_name`

the firmware image name

Description

Cache firmware in kernel memory so that drivers can use it when system isn't ready for them to request firmware image from userspace. Once it returns successfully, driver can use request_firmware or its nowait version to get the cached firmware without any interacting with userspace

Return 0 if the firmware image has been cached successfully Return !0 otherwise

uncache_firmware

LINUX

Name

`uncache_firmware` — remove one cached firmware image

Synopsis

```
int uncache_firmware (const char * fw_name);
```

Arguments

`fw_name`

the firmware image name

Description

Uncache one firmware image which has been cached successfully before.

Return 0 if the firmware cache has been removed successfully Return !0 otherwise

transport_class_register

LINUX

Name

`transport_class_register` — register an initial transport class

Synopsis

```
int transport_class_register (struct transport_class *
tclass);
```

Arguments

tclass

a pointer to the transport class structure to be initialised

Description

The transport class contains an embedded class which is used to identify it. The caller should initialise this structure with zeros and then generic class must have been initialised with the actual transport class unique name. There's a macro DECLARE_TRANSPORT_CLASS to do this (declared classes still must be registered).

Returns 0 on success or error on failure.

transport_class_unregister

LINUX

Kernel Hackers ManualApril 2016

Name

transport_class_unregister — unregister a previously registered class

Synopsis

```
void transport_class_unregister (struct transport_class *
tclass);
```

Arguments

tclass

The transport class to unregister

Description

Must be called prior to deallocating the memory for the transport class.

anon_transport_class_register

LINUX

Kernel Hackers ManualApril 2016

Name

anon_transport_class_register — register an anonymous class

Synopsis

```
int anon_transport_class_register (struct anon_transport_class
* atc);
```

Arguments

atc

The anon transport class to register

Description

The anonymous transport class contains both a transport class and a container. The idea of an anonymous class is that it never actually has any device attributes associated with it (and thus saves on container storage). So it can only be used for triggering events. Use prezero and then use `DECLARE_ANON_TRANSPORT_CLASS` to initialise the anon transport class storage.

anon_transport_class_unregister

LINUX

Kernel Hackers ManualApril 2016

Name

`anon_transport_class_unregister` — unregister an anon class

Synopsis

```
void anon_transport_class_unregister (struct
anon_transport_class * atc);
```

Arguments

`atc`

Pointer to the anon transport class to unregister

Description

Must be called prior to deallocating the memory for the anon transport class.

transport_setup_device

LINUX

Kernel Hackers ManualApril 2016

Name

`transport_setup_device` — declare a new dev for transport class association but don't make it visible yet.

Synopsis

```
void transport_setup_device (struct device * dev);
```

Arguments

dev

the generic device representing the entity being added

Description

Usually, dev represents some component in the HBA system (either the HBA itself or a device remote across the HBA bus). This routine is simply a trigger point to see if any set of transport classes wishes to associate with the added device. This allocates storage for the class device and initialises it, but does not yet add it to the system or add attributes to it (you do this with transport_add_device). If you have no need for a separate setup and add operations, use transport_register_device (see transport_class.h).

transport_add_device

LINUX

Kernel Hackers ManualApril 2016

Name

`transport_add_device` — declare a new dev for transport class association

Synopsis

```
void transport_add_device (struct device * dev);
```

Arguments

dev

the generic device representing the entity being added

Description

Usually, dev represents some component in the HBA system (either the HBA itself or a device remote across the HBA bus). This routine is simply a trigger point used to add the device to the system and register attributes for it.

transport_configure_device

LINUX

Name

`transport_configure_device` — configure an already set up device

Synopsis

```
void transport_configure_device (struct device * dev);
```

Arguments

dev

generic device representing device to be configured

Description

The idea of configure is simply to provide a point within the setup process to allow the transport class to extract information from a device after it has been setup. This is used in SCSI because we have to have a setup device to begin using the HBA, but after we send the initial inquiry, we use configure to extract the device parameters. The device need not have been added to be configured.

transport_remove_device

LINUX

Name

`transport_remove_device` — remove the visibility of a device

Synopsis

```
void transport_remove_device (struct device * dev);
```

Arguments

dev

generic device to remove

Description

This call removes the visibility of the device (to the user from sysfs), but does not destroy it. To eliminate a device entirely you must also call transport_destroy_device. If you don't need to do remove and destroy as separate operations, use `transport_unregister_device` (see transport_class.h) which will perform both calls for you.

transport_destroy_device

LINUX

Kernel Hackers ManualApril 2016

Name

`transport_destroy_device` — destroy a removed device

Synopsis

```
void transport_destroy_device (struct device * dev);
```

Arguments

dev

device to eliminate from the transport class.

Description

This call triggers the elimination of storage associated with the transport classdev. Note: all it really does is relinquish a reference to the classdev. The memory will not be freed until the last reference goes to zero. Note also that the classdev retains a reference count on dev, so dev too will remain for as long as the transport class device remains around.

sysdev_driver_register

LINUX

Kernel Hackers ManualApril 2016

Name

`sysdev_driver_register` — Register auxiliary driver

Synopsis

```
int sysdev_driver_register (struct sysdev_class * cls, struct
sysdev_driver * drv);
```

Arguments

cls

Device class driver belongs to.

drv

Driver.

Description

drv is inserted into *cls*->drivers to be called on each operation on devices of that class. The refcount of *cls* is incremented.

sysdev_driver_unregister

LINUX

Kernel Hackers ManualApril 2016

Name

sysdev_driver_unregister — Remove an auxiliary driver.

Synopsis

```
void sysdev_driver_unregister (struct sysdev_class * cls,
struct sysdev_driver * drv);
```

Arguments

cls

Class driver belongs to.

drv

Driver.

sysdev_register

LINUX

Kernel Hackers ManualApril 2016

Name

`sysdev_register` — add a system device to the tree

Synopsis

```
int sysdev_register (struct sys_device * sysdev);
```

Arguments

`sysdev`

device in question

platform_device_register_simple

LINUX

Kernel Hackers ManualApril 2016

Name

`platform_device_register_simple` — add a platform-level device and its resources

Synopsis

```
struct platform_device * platform_device_register_simple
(const char * name, int id, const struct resource * res,
unsigned int num);
```

Arguments

name

base name of the device we're adding

id

instance id

res

set of resources that needs to be allocated for the device

num

number of resources

Description

This function creates a simple platform device that requires minimal resource and memory management. Canned release function freeing memory allocated for the device allows drivers using such devices to be unloaded without waiting for the last reference to the device to be dropped.

This interface is primarily intended for use with legacy drivers which probe hardware directly. Because such drivers create sysfs device nodes themselves, rather than letting system infrastructure handle such device enumeration tasks, they don't fully conform to the Linux driver model. In particular, when such drivers are built as modules, they can't be "hotplugged".

Returns struct platform_device pointer on success, or `ERR_PTR` on error.

platform_device_register_data

LINUX

Kernel Hackers ManualApril 2016

Name

`platform_device_register_data` — add a platform-level device with platform-specific data

Synopsis

```
struct platform_device * platform_device_register_data (struct
device * parent, const char * name, int id, const void * data,
size_t size);
```

Arguments

parent

parent device for the device we're adding

name

base name of the device we're adding

id

instance id

data

platform specific data for this platform device

size

size of platform specific data

Description

This function creates a simple platform device that requires minimal resource and memory management. Canned release function freeing memory allocated for the device allows drivers using such devices to be unloaded without waiting for the last reference to the device to be dropped.

Returns struct platform_device pointer on success, or `ERR_PTR` on error.

platform_get_resource

LINUX

Kernel Hackers ManualApril 2016

Name

`platform_get_resource` — get a resource for a device

Synopsis

```
struct resource * platform_get_resource (struct
platform_device * dev, unsigned int type, unsigned int num);
```

Arguments

dev

platform device

type

resource type

num

resource index

platform_get_irq

LINUX

Kernel Hackers ManualApril 2016

Name

`platform_get_irq` — get an IRQ for a device

Synopsis

```
int platform_get_irq (struct platform_device * dev, unsigned
int num);
```

Arguments

dev

platform device

num

IRQ number index

platform_get_resource_byname

LINUX

Name

`platform_get_resource_byname` — get a resource for a device by name

Synopsis

```
struct resource * platform_get_resource_byname (struct
platform_device * dev, unsigned int type, const char * name);
```

Arguments

dev

platform device

type

resource type

name

resource name

platform_get_irq_byname

LINUX

Kernel Hackers ManualApril 2016

Name

`platform_get_irq_byname` — get an IRQ for a device

Synopsis

```
int platform_get_irq_byname (struct platform_device * dev,
const char * name);
```

Arguments

dev

platform device

name

IRQ name

platform_add_devices

LINUX

Kernel Hackers ManualApril 2016

Name

`platform_add_devices` — add a numbers of platform devices

Synopsis

```
int platform_add_devices (struct platform_device ** devs, int
num);
```

Arguments

devs

array of platform devices to add

num

number of platform devices in array

platform_device_put

LINUX

Kernel Hackers ManualApril 2016

Name

`platform_device_put` — destroy a platform device

Synopsis

```
void platform_device_put (struct platform_device * pdev);
```

Arguments

pdev

platform device to free

Description

Free all memory associated with a platform device. This function must _only_ be externally called in error cases. All other usage is a bug.

platform_device_alloc

LINUX

Kernel Hackers ManualApril 2016

Name

`platform_device_alloc` — create a platform device

Synopsis

```
struct platform_device * platform_device_alloc (const char *
name, int id);
```

Arguments

name

base name of the device we're adding

id

instance id

Description

Create a platform device object which can have other objects attached to it, and which will have attached objects freed when it is released.

platform_device_add_resources

LINUX

Kernel Hackers ManualApril 2016

Name

`platform_device_add_resources` — add resources to a platform device

Synopsis

```
int platform_device_add_resources (struct platform_device *
pdev, const struct resource * res, unsigned int num);
```

Arguments

pdev

platform device allocated by platform_device_alloc to add resources to

res

set of resources that needs to be allocated for the device

num

number of resources

Description

Add a copy of the resources to the platform device. The memory associated with the resources will be freed when the platform device is released.

platform_device_add_data

LINUX

Kernel Hackers ManualApril 2016

Name

`platform_device_add_data` — add platform-specific data to a platform device

Synopsis

```
int platform_device_add_data (struct platform_device * pdev,
const void * data, size_t size);
```

Arguments

pdev

platform device allocated by platform_device_alloc to add resources to

data

platform specific data for this platform device

size

size of platform specific data

Description

Add a copy of platform specific data to the platform device's platform_data pointer. The memory associated with the platform data will be freed when the platform device is released.

platform_device_add

LINUX

Kernel Hackers ManualApril 2016

Name

`platform_device_add` — add a platform device to device hierarchy

Synopsis

```
int platform_device_add (struct platform_device * pdev);
```

Arguments

`pdev`

platform device we're adding

Description

This is part 2 of `platform_device_register`, though may be called separately _iff_ pdev was allocated by `platform_device_alloc`.

platform_device_del

LINUX

Name

`platform_device_del` — remove a platform-level device

Synopsis

```
void platform_device_del (struct platform_device * pdev);
```

Arguments

pdev

platform device we're removing

Description

Note that this function will also release all memory- and port-based resources owned by the device (*dev*->resource). This function must _only_ be externally called in error cases. All other usage is a bug.

platform_device_register

LINUX

Name

`platform_device_register` — add a platform-level device

Synopsis

```
int platform_device_register (struct platform_device * pdev);
```

Arguments

pdev

platform device we're adding

platform_device_unregister

LINUX

Kernel Hackers ManualApril 2016

Name

`platform_device_unregister` — unregister a platform-level device

Synopsis

```
void platform_device_unregister (struct platform_device *
pdev);
```

Arguments

pdev

platform device we're unregistering

Description

Unregistration is done in 2 steps. First we release all resources and remove it from the subsystem, then we drop reference count by calling `platform_device_put`.

platform_device_register_resndata

LINUX

Kernel Hackers ManualApril 2016

Name

`platform_device_register_resndata` — add a platform-level device with resources and platform-specific data

Synopsis

```
struct platform_device * platform_device_register_resndata
(struct device * parent, const char * name, int id, const
struct resource * res, unsigned int num, const void * data,
size_t size);
```

Arguments

parent

parent device for the device we're adding

name

base name of the device we're adding

id

instance id

res

set of resources that needs to be allocated for the device

num

number of resources

data

platform specific data for this platform device

size

size of platform specific data

Description

Returns struct platform_device pointer on success, or ERR_PTR on error.

platform_driver_register

LINUX

Kernel Hackers ManualApril 2016

Name

platform_driver_register — register a driver for platform-level devices

Synopsis

```
int platform_driver_register (struct platform_driver * drv);
```

Arguments

drv

platform driver structure

platform_driver_unregister

LINUX

Kernel Hackers ManualApril 2016

Name

`platform_driver_unregister` — unregister a driver for platform-level devices

Synopsis

```
void platform_driver_unregister (struct platform_driver * drv);
```

Arguments

drv

platform driver structure

platform_driver_probe

LINUX

Name

`platform_driver_probe` — register driver for non-hotpluggable device

Synopsis

```
int platform_driver_probe (struct platform_driver * drv, int
(*probe) (struct platform_device *));
```

Arguments

drv

platform driver structure

probe

the driver probe routine, probably from an __init section

Description

Use this instead of `platform_driver_register` when you know the device is not hotpluggable and has already been registered, and you want to remove its run-once `probe` infrastructure from memory after the driver has bound to the device.

One typical use for this would be with drivers for controllers integrated into system-on-chip processors, where the controller devices have been configured as part of board setup.

Returns zero if the driver registered and bound to a device, else returns a negative error code and with the driver not registered.

platform_create_bundle

LINUX

Kernel Hackers ManualApril 2016

Name

`platform_create_bundle` — register driver and create corresponding device

Synopsis

```
struct platform_device * platform_create_bundle (struct
platform_driver * driver, int (*probe) (struct platform_device
*), struct resource * res, unsigned int n_res, const void *
data, size_t size);
```

Arguments

driver

platform driver structure

probe

the driver probe routine, probably from an __init section

res

set of resources that needs to be allocated for the device

n_res

number of resources

data

platform specific data for this platform device

size

size of platform specific data

Description

Use this in legacy-style modules that probe hardware directly and register a single platform device and corresponding platform driver.

Returns struct platform_device pointer on success, or `ERR_PTR` on error.

bus_for_each_dev

LINUX

Kernel Hackers ManualApril 2016

Name

`bus_for_each_dev` — device iterator.

Synopsis

```
int bus_for_each_dev (struct bus_type * bus, struct device *
start, void * data, int (*fn) (struct device *, void *));
```

Arguments

`bus`

bus type.

`start`

device to start iterating from.

`data`

data for the callback.

fn

function to be called for each device.

Description

Iterate over *bus*'s list of devices, and call *fn* for each, passing it *data*. If *start* is not NULL, we use that device to begin iterating from.

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

NOTE

The device that returns a non-zero value is not retained in any way, nor is its refcount incremented. If the caller needs to retain this data, it should do so, and increment the reference count in the supplied callback.

bus_find_device

LINUX

Kernel Hackers ManualApril 2016

Name

bus_find_device — device iterator for locating a particular device.

Synopsis

```
struct device * bus_find_device (struct bus_type * bus, struct
device * start, void * data, int (*match) (struct device *dev,
void *data));
```

Arguments

bus

bus type

start

Device to begin with

data

Data to pass to match function

match

Callback function to check device

Description

This is similar to the bus_for_each_dev function above, but it returns a reference to a device that is 'found' for later use, as determined by the *match* callback.

The callback should return 0 if the device doesn't match and non-zero if it does. If the callback returns non-zero, this function will return to the caller and not iterate over any more devices.

bus_find_device_by_name

LINUX

Kernel Hackers ManualApril 2016

Name

bus_find_device_by_name — device iterator for locating a particular device of a specific name

Synopsis

```
struct device * bus_find_device_by_name (struct bus_type *
bus, struct device * start, const char * name);
```

Arguments

bus

bus type

start

Device to begin with

name

name of the device to match

Description

This is similar to the `bus_find_device` function above, but it handles searching by a name automatically, no need to write another strcmp matching function.

bus_for_each_drv

LINUX

Kernel Hackers ManualApril 2016

Name

`bus_for_each_drv` — driver iterator

Synopsis

```
int bus_for_each_drv (struct bus_type * bus, struct
device_driver * start, void * data, int (*fn) (struct
device_driver *, void *));
```

Arguments

bus

bus we're dealing with.

start

driver to start iterating on.

data

data to pass to the callback.

fn

function to call for each driver.

Description

This is nearly identical to the device iterator above. We iterate over each driver that belongs to *bus*, and call *fn* for each. If *fn* returns anything but 0, we break out and return it. If *start* is not NULL, we use it as the head of the list.

NOTE

we don't return the driver that returns a non-zero value, nor do we leave the reference count incremented for that driver. If the caller needs to know that info, it must set it in the callback. It must also be sure to increment the refcount so it doesn't disappear before returning to the caller.

bus_rescan_devices

LINUX

Kernel Hackers ManualApril 2016

Name

bus_rescan_devices — rescan devices on the bus for possible drivers

Synopsis

```
int bus_rescan_devices (struct bus_type * bus);
```

Arguments

bus

the bus to scan.

Description

This function will look for devices on the bus with no driver attached and rescan it against existing drivers to see if it matches any by calling device_attach for the unbound devices.

device_reprobe

LINUX

Name

`device_reprobe` — remove driver for a device and probe for a new driver

Synopsis

```
int device_reprobe (struct device * dev);
```

Arguments

`dev`

the device to reprobe

Description

This function detaches the attached driver (if any) for the given device and restarts the driver probing process. It is intended to use if probing criteria changed during a devices lifetime and driver attachment should change accordingly.

bus_register

LINUX

Name

`bus_register` — register a bus with the system.

Synopsis

```
int bus_register (struct bus_type * bus);
```

Arguments

bus

bus.

Description

Once we have that, we registered the bus with the kobject infrastructure, then register the children subsystems it has: the devices and drivers that belong to the bus.

bus_unregister

LINUX

Kernel Hackers ManualApril 2016

Name

bus_unregister — remove a bus from the system

Synopsis

```
void bus_unregister (struct bus_type * bus);
```

Arguments

bus

bus.

Description

Unregister the child subsystems and the bus itself. Finally, we call `bus_put` to release the refcount

2.3. Device Drivers Power Management

dpm_resume_noirq

LINUX

Kernel Hackers ManualApril 2016

Name

`dpm_resume_noirq` — Execute “early resume” callbacks for non-sysdev devices.

Synopsis

```
void dpm_resume_noirq (pm_message_t state);
```

Arguments

state

PM transition of the system being carried out.

Description

Call the “noirq” resume handlers for all devices marked as DPM_OFF_IRQ and enable device drivers to receive interrupts.

dpm_resume_end

LINUX

Kernel Hackers ManualApril 2016

Name

dpm_resume_end — Execute “resume” callbacks and complete system transition.

Synopsis

```
void dpm_resume_end (pm_message_t state);
```

Arguments

state

PM transition of the system being carried out.

Description

Execute “resume” callbacks for all devices and complete the PM transition of the system.

dpm_suspend_noirq

LINUX

Kernel Hackers ManualApril 2016

Name

`dpm_suspend_noirq` — Execute “late suspend” callbacks for non-sysdev devices.

Synopsis

```
int dpm_suspend_noirq (pm_message_t state);
```

Arguments

`state`

PM transition of the system being carried out.

Description

Prevent device drivers from receiving interrupts and call the “noirq” suspend handlers for all non-sysdev devices.

dpm_suspend_start

LINUX

Kernel Hackers ManualApril 2016

Name

`dpm_suspend_start` — Prepare devices for PM transition and suspend them.

Synopsis

```
int dpm_suspend_start (pm_message_t state);
```

Arguments

state

PM transition of the system being carried out.

Description

Prepare all non-sysdev devices for system PM transition and execute “suspend” callbacks for them.

device_pm_wait_for_dev

LINUX

Name

`device_pm_wait_for_dev` — Wait for suspend/resume of a device to complete.

Synopsis

```
int device_pm_wait_for_dev (struct device * subordinate,
struct device * dev);
```

Arguments

`subordinate`

Device that needs to wait for *`dev`*.

`dev`

Device to wait for.

dpm_for_each_dev

LINUX

Name

`dpm_for_each_dev` — device iterator.

Synopsis

```
void dpm_for_each_dev (void * data, void (*fn) (struct device
*, void *));
```

Arguments

data

data for the callback.

fn

function to be called for each device.

Description

Iterate over devices in dpm_list, and call *fn* for each device, passing it *data*.

2.4. Device Drivers ACPI Support

acpi_bus_register_driver

LINUX

Kernel Hackers ManualApril 2016

Name

acpi_bus_register_driver — register a driver with the ACPI bus

Synopsis

```
int acpi_bus_register_driver (struct acpi_driver * driver);
```

Arguments

driver

driver being registered

Description

Registers a driver with the ACPI bus. Searches the namespace for all devices that match the driver's criteria and binds. Returns zero for success or a negative error status for failure.

acpi_bus_unregister_driver

LINUX

Kernel Hackers ManualApril 2016

Name

`acpi_bus_unregister_driver` — unregisters a driver with the APIC bus

Synopsis

```
void acpi_bus_unregister_driver (struct acpi_driver * driver);
```

Arguments

driver

driver to unregister

Description

Unregisters a driver with the ACPI bus. Searches the namespace for all devices that match the driver's criteria and unbinds.

acpi_bus_driver_init

LINUX

Kernel Hackers ManualApril 2016

Name

`acpi_bus_driver_init` — add a device to a driver

Synopsis

```
int acpi_bus_driver_init (struct acpi_device * device, struct
acpi_driver * driver);
```

Arguments

device

the device to add and initialize

driver

driver for the device

Description

Used to initialize a device via its device driver. Called whenever a driver is bound to a device. Invokes the driver's `add` ops.

2.5. Device drivers PnP support

pnp_register_protocol

LINUX

Kernel Hackers ManualApril 2016

Name

`pnp_register_protocol` — adds a pnp protocol to the pnp layer

Synopsis

```
int pnp_register_protocol (struct pnp_protocol * protocol);
```

Arguments

`protocol`

pointer to the corresponding pnp_protocol structure

Ex protocols

ISAPNP, PNPBIOS, etc

pnp_unregister_protocol

LINUX

Kernel Hackers ManualApril 2016

Name

pnp_unregister_protocol — removes a pnp protocol from the pnp layer

Synopsis

```
void pnp_unregister_protocol (struct pnp_protocol * protocol);
```

Arguments

protocol

pointer to the corresponding pnp_protocol structure

pnp_request_card_device

LINUX

Kernel Hackers ManualApril 2016

Name

pnp_request_card_device — Searches for a PnP device under the specified card

Synopsis

```
struct pnp_dev * pnp_request_card_device (struct pnp_card_link
* clink, const char * id, struct pnp_dev * from);
```

Arguments

clink

pointer to the card link, cannot be NULL

id

pointer to a PnP ID structure that explains the rules for finding the device

from

Starting place to search from. If NULL it will start from the beginning.

pnp_release_card_device

LINUX

Kernel Hackers ManualApril 2016

Name

pnp_release_card_device — call this when the driver no longer needs the device

Synopsis

```
void pnp_release_card_device (struct pnp_dev * dev);
```

Arguments

dev

pointer to the PnP device structure

pnp_register_card_driver

LINUX

Kernel Hackers ManualApril 2016

Name

pnp_register_card_driver — registers a PnP card driver with the PnP Layer

Synopsis

```
int pnp_register_card_driver (struct pnp_card_driver * drv);
```

Arguments

drv

pointer to the driver to register

pnp_unregister_card_driver

LINUX

Name

`pnp_unregister_card_driver` — unregisters a PnP card driver from the PnP Layer

Synopsis

```
void pnp_unregister_card_driver (struct pnp_card_driver *
drv);
```

Arguments

drv

pointer to the driver to unregister

pnp_add_id

LINUX

Name

`pnp_add_id` — adds an EISA id to the specified device

Synopsis

```
struct pnp_id * pnp_add_id (struct pnp_dev * dev, const char *
id);
```

Arguments

dev

pointer to the desired device

id

pointer to an EISA id string

pnp_start_dev

LINUX

Kernel Hackers ManualApril 2016

Name

pnp_start_dev — low-level start of the PnP device

Synopsis

```
int pnp_start_dev (struct pnp_dev * dev);
```

Arguments

dev

pointer to the desired device

Description

assumes that resources have already been allocated

pnp_stop_dev

LINUX

Kernel Hackers ManualApril 2016

Name

`pnp_stop_dev` — low-level disable of the PnP device

Synopsis

```
int pnp_stop_dev (struct pnp_dev * dev);
```

Arguments

dev

pointer to the desired device

Description

does not free resources

pnp_activate_dev

LINUX

Kernel Hackers ManualApril 2016

Name

pnp_activate_dev — activates a PnP device for use

Synopsis

```
int pnp_activate_dev (struct pnp_dev * dev);
```

Arguments

dev

pointer to the desired device

Description

does not validate or set resources so be careful.

pnp_disable_dev

LINUX

Kernel Hackers ManualApril 2016

Name

pnp_disable_dev — disables device

Synopsis

```
int pnp_disable_dev (struct pnp_dev * dev);
```

Arguments

dev

pointer to the desired device

Description

inform the correct pnp protocol so that resources can be used by other devices

pnp_is_active

LINUX

Kernel Hackers ManualApril 2016

Name

`pnp_is_active` — Determines if a device is active based on its current resources

Synopsis

```
int pnp_is_active (struct pnp_dev * dev);
```

Arguments

dev

pointer to the desired PnP device

2.6. Userspace IO devices

uio_event_notify

LINUX

Kernel Hackers ManualApril 2016

Name

`uio_event_notify` — trigger an interrupt event

Synopsis

```
void uio_event_notify (struct uio_info * info);
```

Arguments

info

UIO device capabilities

__uio_register_device

LINUX

Kernel Hackers ManualApril 2016

Name

__uio_register_device — register a new userspace IO device

Synopsis

```
int __uio_register_device (struct module * owner, struct
device * parent, struct uio_info * info);
```

Arguments

owner

module that creates the new device

parent

parent device

info

UIO device capabilities

Description

returns zero on success or a negative error code.

uio_unregister_device

LINUX

Kernel Hackers ManualApril 2016

Name

`uio_unregister_device` — unregister a industrial IO device

Synopsis

```
void uio_unregister_device (struct uio_info * info);
```

Arguments

info

UIO device capabilities

struct uio_mem

LINUX

Kernel Hackers ManualApril 2016

Name

`struct uio_mem` — description of a UIO memory region

Synopsis

```
struct uio_mem {
```

```
  const char * name;
  unsigned long addr;
  unsigned long size;
  int memtype;
  void __iomem * internal_addr;
  struct uio_map * map;
};
```

Members

name

name of the memory region for identification

addr

address of the device's memory

size

size of IO

memtype

type of memory addr points to

internal_addr

ioremap-ped version of addr, for driver internal use

map

for use by the UIO core only.

struct uio_port

LINUX

Name

`struct uio_port` — description of a UIO port region

Synopsis

```
struct uio_port {
  const char * name;
  unsigned long start;
  unsigned long size;
  int porttype;
  struct uio_portio * portio;
};
```

Members

name

name of the port region for identification

start

start of port region

size

size of port region

porttype

type of port (see UIO_PORT_* below)

portio

for use by the UIO core only.

struct uio_info

LINUX

Kernel Hackers ManualApril 2016

Name

`struct uio_info` — UIO device capabilities

Synopsis

```
struct uio_info {
  struct uio_device * uio_dev;
  const char * name;
  const char * version;
  struct uio_mem mem[MAX_UIO_MAPS];
  struct uio_port port[MAX_UIO_PORT_REGIONS];
  long irq;
  unsigned long irq_flags;
  void * priv;
  irqreturn_t (* handler) (int irq, struct uio_info *dev_info);
  int (* mmap) (struct uio_info *info, struct vm_area_struct *vma);
  int (* open) (struct uio_info *info, struct inode *inode);
  int (* release) (struct uio_info *info, struct inode *inode);
  int (* irqcontrol) (struct uio_info *info, s32 irq_on);
};
```

Members

uio_dev

the UIO device this info belongs to

name

device name

version

device driver version

mem[MAX_UIO_MAPS]

: list of mappable memory regions, size==0 for end of list

port[MAX_UIO_PORT_REGIONS]

: list of port regions, size==0 for end of list

irq

: interrupt number or UIO_IRQ_CUSTOM

irq_flags

: flags for `request_irq`

priv

: optional private data

handler

: the device's irq handler

mmap

: mmap operation for this uio device

open

: open operation for this uio device

release

: release operation for this uio device

irqcontrol

: disable/enable irqs when 0/1 is written to /dev/uioX

Chapter 3. Parallel Port Devices

parport_yield

LINUX

Kernel Hackers ManualApril 2016

Name

`parport_yield` — relinquish a parallel port temporarily

Synopsis

```
int parport_yield (struct pardevice * dev);
```

Arguments

dev

a device on the parallel port

Description

This function relinquishes the port if it would be helpful to other drivers to do so. Afterwards it tries to reclaim the port using `parport_claim`, and the return value is the same as for `parport_claim`. If it fails, the port is left unclaimed and it is the driver's responsibility to reclaim the port.

The `parport_yield` and `parport_yield_blocking` functions are for marking points in the driver at which other drivers may claim the port and use their devices. Yielding the port is similar to releasing it and reclaiming it, but is more efficient because no action is taken if there are no other devices needing the port. In fact, nothing is done even if there are other devices waiting but the current device is still

within its “timeslice”. The default timeslice is half a second, but it can be adjusted via the /proc interface.

parport_yield_blocking

LINUX

Kernel Hackers ManualApril 2016

Name

`parport_yield_blocking` — relinquish a parallel port temporarily

Synopsis

```
int parport_yield_blocking (struct pardevice * dev);
```

Arguments

dev

a device on the parallel port

Description

This function relinquishes the port if it would be helpful to other drivers to do so. Afterwards it tries to reclaim the port using `parport_claim_or_block`, and the return value is the same as for `parport_claim_or_block`.

parport_wait_event

LINUX

Kernel Hackers ManualApril 2016

Name

`parport_wait_event` — wait for an event on a parallel port

Synopsis

```
int parport_wait_event (struct parport * port, signed long
timeout);
```

Arguments

port

port to wait on

timeout

time to wait (in jiffies)

Description

This function waits for up to *timeout* jiffies for an interrupt to occur on a parallel port. If the port timeout is set to zero, it returns immediately.

If an interrupt occurs before the timeout period elapses, this function returns zero immediately. If it times out, it returns one. An error code less than zero indicates an error (most likely a pending signal), and the calling code should finish what it's doing as soon as it can.

parport_wait_peripheral

LINUX

Kernel Hackers ManualApril 2016

Name

`parport_wait_peripheral` — wait for status lines to change in 35ms

Synopsis

```
int parport_wait_peripheral (struct parport * port, unsigned
char mask, unsigned char result);
```

Arguments

`port`

port to watch

`mask`

status lines to watch

`result`

desired values of chosen status lines

Description

This function waits until the masked status lines have the desired values, or until 35ms have elapsed (see IEEE 1284-1994 page 24 to 25 for why this value in particular is hardcoded). The *`mask`* and *`result`* parameters are bitmasks, with the bits defined by the constants in parport.h: `PARPORT_STATUS_BUSY`, and so on.

The port is polled quickly to start off with, in anticipation of a fast response from the peripheral. This fast polling time is configurable (using /proc), and defaults to

500usec. If the timeout for this port (see `parport_set_timeout`) is zero, the fast polling time is 35ms, and this function does not call `schedule`.

If the timeout for this port is non-zero, after the fast polling fails it uses `parport_wait_event` to wait for up to 10ms, waking up if an interrupt occurs.

parport_negotiate

LINUX

Kernel Hackers ManualApril 2016

Name

`parport_negotiate` — negotiate an IEEE 1284 mode

Synopsis

```
int parport_negotiate (struct parport * port, int mode);
```

Arguments

port

port to use

mode

mode to negotiate to

Description

Use this to negotiate to a particular IEEE 1284 transfer mode. The *mode* parameter should be one of the constants in parport.h starting `IEEE1284_MODE_xxx`.

The return value is 0 if the peripheral has accepted the negotiation to the mode specified, -1 if the peripheral is not IEEE 1284 compliant (or not present), or 1 if the peripheral has rejected the negotiation.

parport_write

LINUX

Kernel Hackers ManualApril 2016

Name

`parport_write` — write a block of data to a parallel port

Synopsis

```
ssize_t parport_write (struct parport * port, const void *
buffer, size_t len);
```

Arguments

port

port to write to

buffer

data buffer (in kernel space)

len

number of bytes of data to transfer

Description

This will write up to *len* bytes of *buffer* to the port specified, using the IEEE 1284 transfer mode most recently negotiated to (using parport_negotiate), as long as that mode supports forward transfers (host to peripheral).

It is the caller's responsibility to ensure that the first *len* bytes of *buffer* are valid.

This function returns the number of bytes transferred (if zero or positive), or else an error code.

parport_read

LINUX

Kernel Hackers ManualApril 2016

Name

parport_read — read a block of data from a parallel port

Synopsis

```
ssize_t parport_read (struct parport * port, void * buffer,
size_t len);
```

Arguments

port

port to read from

buffer

data buffer (in kernel space)

len

number of bytes of data to transfer

Description

This will read up to *len* bytes of *buffer* to the port specified, using the IEEE 1284 transfer mode most recently negotiated to (using parport_negotiate), as long as that mode supports reverse transfers (peripheral to host).

It is the caller's responsibility to ensure that the first *len* bytes of *buffer* are available to write to.

This function returns the number of bytes transferred (if zero or positive), or else an error code.

parport_set_timeout

LINUX

Kernel Hackers ManualApril 2016

Name

parport_set_timeout — set the inactivity timeout for a device

Synopsis

```
long parport_set_timeout (struct pardevice * dev, long
inactivity);
```

Arguments

dev

device on a port

inactivity

inactivity timeout (in jiffies)

Description

This sets the inactivity timeout for a particular device on a port. This affects functions like `parport_wait_peripheral`. The special value 0 means not to call `schedule` while dealing with this device.

The return value is the previous inactivity timeout.

Any callers of `parport_wait_event` for this device are woken up.

parport_register_driver

LINUX

Kernel Hackers ManualApril 2016

Name

`parport_register_driver` — register a parallel port device driver

Synopsis

```
int parport_register_driver (struct parport_driver * drv);
```

Arguments

drv

structure describing the driver

Description

This can be called by a parallel port device driver in order to receive notifications about ports being found in the system, as well as ports no longer available.

The *drv* structure is allocated by the caller and must not be deallocated until after calling parport_unregister_driver.

The driver's attach function may block. The port that attach is given will be valid for the duration of the callback, but if the driver wants to take a copy of the pointer it must call parport_get_port to do so. Calling parport_register_device on that port will do this for you.

The driver's detach function may block. The port that detach is given will be valid for the duration of the callback, but if the driver wants to take a copy of the pointer it must call parport_get_port to do so.

Returns 0 on success. Currently it always succeeds.

parport_unregister_driver

LINUX

Kernel Hackers ManualApril 2016

Name

parport_unregister_driver — deregister a parallel port device driver

Synopsis

```
void parport_unregister_driver (struct parport_driver * drv);
```

Arguments

drv

structure describing the driver that was given to `parport_register_driver`

Description

This should be called by a parallel port device driver that has registered itself using `parport_register_driver` when it is about to be unloaded.

When it returns, the driver's `attach` routine will no longer be called, and for each port that `attach` was called for, the `detach` routine will have been called.

All the driver's `attach` and `detach` calls are guaranteed to have finished by the time this function returns.

parport_get_port

LINUX

Kernel Hackers ManualApril 2016

Name

`parport_get_port` — increment a port's reference count

Synopsis

```
struct parport * parport_get_port (struct parport * port);
```

Arguments

port

the port

Description

This ensures that a struct parport pointer remains valid until the matching `parport_put_port` call.

parport_put_port

LINUX

Kernel Hackers ManualApril 2016

Name

`parport_put_port` — decrement a port's reference count

Synopsis

```
void parport_put_port (struct parport * port);
```

Arguments

port

the port

Description

This should be called once for each call to `parport_get_port`, once the port is no longer needed.

parport_register_port

LINUX

Kernel Hackers ManualApril 2016

Name

`parport_register_port` — register a parallel port

Synopsis

```
struct parport * parport_register_port (unsigned long base,
int irq, int dma, struct parport_operations * ops);
```

Arguments

base

base I/O address

irq

IRQ line

dma

DMA channel

ops

pointer to the port driver's port operations structure

Description

When a parallel port (lowlevel) driver finds a port that should be made available to parallel port device drivers, it should call `parport_register_port`. The *`base`*, *`irq`*, and *`dma`* parameters are for the convenience of port drivers, and for ports where they aren't meaningful needn't be set to anything special. They can be altered afterwards by adjusting the relevant members of the parport structure that is returned and represents the port. They should not be tampered with after calling parport_announce_port, however.

If there are parallel port device drivers in the system that have registered themselves using `parport_register_driver`, they are not told about the port at this time; that is done by `parport_announce_port`.

The *`ops`* structure is allocated by the caller, and must not be deallocated before calling `parport_remove_port`.

If there is no memory to allocate a new parport structure, this function will return `NULL`.

parport_announce_port

LINUX

Kernel Hackers ManualApril 2016

Name

`parport_announce_port` — tell device drivers about a parallel port

Synopsis

```
void parport_announce_port (struct parport * port);
```

Arguments

port

parallel port to announce

Description

After a port driver has registered a parallel port with parport_register_port, and performed any necessary initialisation or adjustments, it should call `parport_announce_port` in order to notify all device drivers that have called `parport_register_driver`. Their `attach` functions will be called, with *port* as the parameter.

parport_remove_port

LINUX

Kernel Hackers ManualApril 2016

Name

`parport_remove_port` — deregister a parallel port

Synopsis

```
void parport_remove_port (struct parport * port);
```

Arguments

port

parallel port to deregister

Description

When a parallel port driver is forcibly unloaded, or a parallel port becomes inaccessible, the port driver must call this function in order to deal with device drivers that still want to use it.

The parport structure associated with the port has its operations structure replaced with one containing 'null' operations that return errors or just don't do anything.

Any drivers that have registered themselves using `parport_register_driver` are notified that the port is no longer accessible by having their `detach` routines called with *`port`* as the parameter.

parport_register_device

LINUX

Kernel Hackers ManualApril 2016

Name

`parport_register_device` — register a device on a parallel port

Synopsis

```
struct pardevice * parport_register_device (struct parport *
port, const char * name, int (*pf) (void *), void (*kf) (void
*), void (*irq_func) (void *), int flags, void * handle);
```

Arguments

`port`

port to which the device is attached

name

a name to refer to the device

pf

preemption callback

kf

kick callback (wake-up)

irq_func

interrupt handler

flags

registration flags

handle

data for callback functions

Description

This function, called by parallel port device drivers, declares that a device is connected to a port, and tells the system all it needs to know.

The *name* is allocated by the caller and must not be deallocated until the caller calls *parport_unregister_device* for that device.

The preemption callback function, *pf*, is called when this device driver has claimed access to the port but another device driver wants to use it. It is given *handle* as its parameter, and should return zero if it is willing for the system to release the port to another driver on its behalf. If it wants to keep control of the port it should return non-zero, and no action will be taken. It is good manners for the driver to try to release the port at the earliest opportunity after its preemption callback rejects a preemption attempt. Note that if a preemption callback is happy for preemption to go ahead, there is no need to release the port; it is done automatically. This function may not block, as it may be called from interrupt context. If the device driver does not support preemption, *pf* can be NULL.

The wake-up ("kick") callback function, *kf*, is called when the port is available to be claimed for exclusive access; that is, parport_claim is guaranteed to succeed when called from inside the wake-up callback function. If the driver wants to claim the port it should do so; otherwise, it need not take any action. This function may not block, as it may be called from interrupt context. If the device driver does not want to be explicitly invited to claim the port in this way, *kf* can be NULL.

The interrupt handler, *irq_func*, is called when an interrupt arrives from the parallel port. Note that if a device driver wants to use interrupts it should use parport_enable_irq, and can also check the irq member of the parport structure representing the port.

The parallel port (lowlevel) driver is the one that has called request_irq and whose interrupt handler is called first. This handler does whatever needs to be done to the hardware to acknowledge the interrupt (for PC-style ports there is nothing special to be done). It then tells the IEEE 1284 code about the interrupt, which may involve reacting to an IEEE 1284 event depending on the current IEEE 1284 phase. After this, it calls *irq_func*. Needless to say, *irq_func* will be called from interrupt context, and may not block.

The PARPORT_DEV_EXCL flag is for preventing port sharing, and so should only be used when sharing the port with other device drivers is impossible and would lead to incorrect behaviour. Use it sparingly! Normally, *flags* will be zero.

This function returns a pointer to a structure that represents the device on the port, or NULL if there is not enough memory to allocate space for that structure.

parport_unregister_device

LINUX

Kernel Hackers ManualApril 2016

Name

parport_unregister_device — deregister a device on a parallel port

Synopsis

```
void parport_unregister_device (struct pardevice * dev);
```

Arguments

dev

pointer to structure representing device

Description

This undoes the effect of `parport_register_device`.

parport_find_number

LINUX

Kernel Hackers ManualApril 2016

Name

`parport_find_number` — find a parallel port by number

Synopsis

```
struct parport * parport_find_number (int number);
```

Arguments

number

parallel port number

Description

This returns the parallel port with the specified number, or `NULL` if there is none.

There is an implicit parport_get_port done already; to throw away the reference to the port that parport_find_number gives you, use parport_put_port.

parport_find_base

LINUX

Kernel Hackers ManualApril 2016

Name

parport_find_base — find a parallel port by base address

Synopsis

```
struct parport * parport_find_base (unsigned long base);
```

Arguments

base

base I/O address

Description

This returns the parallel port with the specified base address, or NULL if there is none.

There is an implicit parport_get_port done already; to throw away the reference to the port that parport_find_base gives you, use parport_put_port.

parport_claim

LINUX

Kernel Hackers ManualApril 2016

Name

`parport_claim` — claim access to a parallel port device

Synopsis

```
int parport_claim (struct pardevice * dev);
```

Arguments

dev

pointer to structure representing a device on the port

Description

This function will not block and so can be used from interrupt context. If `parport_claim` succeeds in claiming access to the port it returns zero and the port is available to use. It may fail (returning non-zero) if the port is in use by another driver and that driver is not willing to relinquish control of the port.

parport_claim_or_block

LINUX

Name

`parport_claim_or_block` — claim access to a parallel port device

Synopsis

```
int parport_claim_or_block (struct pardevice * dev);
```

Arguments

dev

pointer to structure representing a device on the port

Description

This behaves like `parport_claim`, but will block if necessary to wait for the port to be free. A return value of 1 indicates that it slept; 0 means that it succeeded without needing to sleep. A negative error code indicates failure.

parport_release

LINUX

Name

`parport_release` — give up access to a parallel port device

Synopsis

```
void parport_release (struct pardevice * dev);
```

Arguments

dev

pointer to structure representing parallel port device

Description

This function cannot fail, but it should not be called without the port claimed. Similarly, if the port is already claimed you should not try claiming it again.

parport_open

LINUX

Kernel Hackers ManualApril 2016

Name

`parport_open` — find a device by canonical device number

Synopsis

```
struct pardevice * parport_open (int devnum, const char *
name);
```

Arguments

devnum

canonical device number

name

name to associate with the device

Description

This function is similar to parport_register_device, except that it locates a device by its number rather than by the port it is attached to.

All parameters except for *devnum* are the same as for parport_register_device. The return value is the same as for parport_register_device.

parport_close

LINUX

Kernel Hackers ManualApril 2016

Name

parport_close — close a device opened with parport_open

Synopsis

```
void parport_close (struct pardevice * dev);
```

Arguments

dev

device to close

Description

This is to `parport_open` as `parport_unregister_device` is to `parport_register_device`.

Chapter 4. Message-based devices

4.1. Fusion message devices

mpt_register

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_register` — Register protocol-specific main callback handler.

Synopsis

```
u8 mpt_register (MPT_CALLBACK cbfunc, MPT_DRIVER_CLASS dclass,
char * func_name);
```

Arguments

`cbfunc`

callback function pointer

`dclass`

Protocol driver's class (`MPT_DRIVER_CLASS` enum value)

`func_name`

-- undescribed --

Description

This routine is called by a protocol-specific driver (SCSI host, LAN, SCSI target) to register its reply callback routine. Each protocol-specific driver must do this before it will be able to use any IOC resources, such as obtaining request frames.

NOTES

The SCSI protocol driver currently calls this routine thrice in order to register separate callbacks; one for “normal” SCSI IO; one for MptScsiTaskMgmt requests; one for Scan/DV requests.

Returns u8 valued “handle” in the range (and S.O.D. order) {N,...,7,6,5,...,1} if successful. A return value of MPT_MAX_PROTOCOL_DRIVERS (including zero!) should be considered an error by the caller.

mpt_deregister

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_deregister` — Deregister a protocol drivers resources.

Synopsis

```
void mpt_deregister (u8 cb_idx);
```

Arguments

`cb_idx`

previously registered callback handle

Description

Each protocol-specific driver should call this routine when its module is unloaded.

mpt_event_register

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_event_register` — Register protocol-specific event callback handler.

Synopsis

```
int mpt_event_register (u8 cb_idx, MPT_EVHANDLER ev_cbfunc);
```

Arguments

`cb_idx`

previously registered (via mpt_register) callback handle

`ev_cbfunc`

callback function

Description

This routine can be called by one or more protocol-specific drivers if/when they choose to be notified of MPT events.

Returns 0 for success.

mpt_event_deregister

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_event_deregister` — Deregister protocol-specific event callback handler

Synopsis

```
void mpt_event_deregister (u8 cb_idx);
```

Arguments

`cb_idx`

previously registered callback handle

Description

Each protocol-specific driver should call this routine when it does not (or can no longer) handle events, or when its module is unloaded.

mpt_reset_register

LINUX

Name

`mpt_reset_register` — Register protocol-specific IOC reset handler.

Synopsis

```
int mpt_reset_register (u8 cb_idx, MPT_RESETHANDLER
reset_func);
```

Arguments

cb_idx

previously registered (via mpt_register) callback handle

reset_func

reset function

Description

This routine can be called by one or more protocol-specific drivers if/when they choose to be notified of IOC resets.

Returns 0 for success.

mpt_reset_deregister

LINUX

Name

`mpt_reset_deregister` — Deregister protocol-specific IOC reset handler.

Synopsis

```
void mpt_reset_deregister (u8 cb_idx);
```

Arguments

`cb_idx`

previously registered callback handle

Description

Each protocol-specific driver should call this routine when it does not (or can no longer) handle IOC reset handling, or when its module is unloaded.

mpt_device_driver_register

LINUX

Name

`mpt_device_driver_register` — Register device driver hooks

Synopsis

```
int mpt_device_driver_register (struct mpt_pci_driver *
dd_cbfunc, u8 cb_idx);
```

Arguments

dd_cbfunc

driver callbacks struct

cb_idx

MPT protocol driver index

mpt_device_driver_deregister

LINUX

Kernel Hackers ManualApril 2016

Name

mpt_device_driver_deregister — DeRegister device driver hooks

Synopsis

```
void mpt_device_driver_deregister (u8 cb_idx);
```

Arguments

cb_idx

MPT protocol driver index

mpt_get_msg_frame

LINUX

Kernel Hackers ManualApril 2016

Name

mpt_get_msg_frame — Obtain an MPT request frame from the pool

Synopsis

```
MPT_FRAME_HDR* mpt_get_msg_frame (u8 cb_idx, MPT_ADAPTER * ioc);
```

Arguments

cb_idx

Handle of registered MPT protocol driver

ioc

Pointer to MPT adapter structure

Description

Obtain an MPT request frame from the pool (of 1024) that are allocated per MPT adapter.

Returns pointer to a MPT request frame or NULL if none are available or IOC is not active.

mpt_put_msg_frame

LINUX

Kernel Hackers ManualApril 2016

Name

mpt_put_msg_frame — Send a protocol-specific MPT request frame to an IOC

Synopsis

```
void mpt_put_msg_frame (u8 cb_idx, MPT_ADAPTER * ioc,
MPT_FRAME_HDR * mf);
```

Arguments

cb_idx

Handle of registered MPT protocol driver

ioc

Pointer to MPT adapter structure

mf

Pointer to MPT request frame

Description

This routine posts an MPT request frame to the request post FIFO of a specific MPT adapter.

mpt_put_msg_frame_hi_pri

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_put_msg_frame_hi_pri` — Send a hi-pri protocol-specific MPT request frame

Synopsis

```
void mpt_put_msg_frame_hi_pri (u8 cb_idx, MPT_ADAPTER * ioc,
MPT_FRAME_HDR * mf);
```

Arguments

cb_idx

Handle of registered MPT protocol driver

ioc

Pointer to MPT adapter structure

mf

Pointer to MPT request frame

Description

Send a protocol-specific MPT request frame to an IOC using hi-priority request queue.

This routine posts an MPT request frame to the request post FIFO of a specific MPT adapter.

mpt_free_msg_frame

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_free_msg_frame` — Place MPT request frame back on FreeQ.

Synopsis

```
void mpt_free_msg_frame (MPT_ADAPTER * ioc, MPT_FRAME_HDR *
mf);
```

Arguments

ioc

Pointer to MPT adapter structure

mf

Pointer to MPT request frame

Description

This routine places a MPT request frame back on the MPT adapter's FreeQ.

mpt_send_handshake_request

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_send_handshake_request` — Send MPT request via doorbell handshake method.

Synopsis

```
int mpt_send_handshake_request (u8 cb_idx, MPT_ADAPTER * ioc,
int reqBytes, u32 * req, int sleepFlag);
```

Arguments

cb_idx

Handle of registered MPT protocol driver

ioc

Pointer to MPT adapter structure

reqBytes

Size of the request in bytes

req

Pointer to MPT request frame

sleepFlag

Use schedule if CAN_SLEEP else use udelay.

Description

This routine is used exclusively to send MptScsiTaskMgmt requests since they are required to be sent via doorbell handshake.

NOTE

It is the callers responsibility to byte-swap fields in the request which are greater than 1 byte in size.

Returns 0 for success, non-zero for failure.

mpt_verify_adapter

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_verify_adapter` — Given IOC identifier, set pointer to its adapter structure.

Synopsis

```
int mpt_verify_adapter (int iocid, MPT_ADAPTER ** iocpp);
```

Arguments

`iocid`

IOC unique identifier (integer)

iocpp

Pointer to pointer to IOC adapter

Description

Given a unique IOC identifier, set pointer to the associated MPT adapter structure.

Returns iocid and sets iocpp if iocid is found. Returns -1 if iocid is not found.

mpt_attach

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_attach` — Install a PCI intelligent MPT adapter.

Synopsis

```
int mpt_attach (struct pci_dev * pdev, const struct
pci_device_id * id);
```

Arguments

pdev

Pointer to pci_dev structure

id

PCI device ID information

Description

This routine performs all the steps necessary to bring the IOC of a MPT adapter to a OPERATIONAL state. This includes registering memory regions, registering the interrupt, and allocating request and reply memory pools.

This routine also pre-fetches the LAN MAC address of a Fibre Channel MPT adapter.

Returns 0 for success, non-zero for failure.

TODO

Add support for polled controllers

mpt_detach

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_detach` — Remove a PCI intelligent MPT adapter.

Synopsis

```
void mpt_detach (struct pci_dev * pdev);
```

Arguments

`pdev`

Pointer to pci_dev structure

mpt_suspend

LINUX

Kernel Hackers ManualApril 2016

Name

mpt_suspend — Fusion MPT base driver suspend routine.

Synopsis

```
int mpt_suspend (struct pci_dev * pdev, pm_message_t state);
```

Arguments

pdev

Pointer to pci_dev structure

state

new state to enter

mpt_resume

LINUX

Name

mpt_resume — Fusion MPT base driver resume routine.

Synopsis

```
int mpt_resume (struct pci_dev * pdev);
```

Arguments

pdev

Pointer to pci_dev structure

mpt_GetIocState

LINUX

Name

mpt_GetIocState — Get the current state of a MPT adapter.

Synopsis

```
u32 mpt_GetIocState (MPT_ADAPTER * ioc, int cooked);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

cooked

Request raw or cooked IOC state

Description

Returns all IOC Doorbell register bits if cooked==0, else just the Doorbell bits in MPI_IOC_STATE_MASK.

mpt_alloc_fw_memory

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_alloc_fw_memory` — allocate firmware memory

Synopsis

```
int mpt_alloc_fw_memory (MPT_ADAPTER * ioc, int size);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

size

total FW bytes

Description

If memory has already been allocated, the same (cached) value is returned.

Return 0 if successful, or non-zero for failure

mpt_free_fw_memory

LINUX

Kernel Hackers ManualApril 2016

Name

mpt_free_fw_memory — free firmware memory

Synopsis

```
void mpt_free_fw_memory (MPT_ADAPTER * ioc);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

Description

If alt_img is NULL, delete from ioc structure. Else, delete a secondary image in same format.

mptbase_sas_persist_operation

LINUX

Kernel Hackers ManualApril 2016

Name

`mptbase_sas_persist_operation` — Perform operation on SAS Persistent Table

Synopsis

```
int mptbase_sas_persist_operation (MPT_ADAPTER * ioc, u8
persist_opcode);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

persist_opcode

see below

Description

MPI_SAS_OP_CLEAR_NOT_PRESENT - Free all persist TargetID mappings for devices not currently present. MPI_SAS_OP_CLEAR_ALL_PERSISTENT - Clear al persist TargetID mappings

NOTE

Don't use not this function during interrupt time.

Returns 0 for success, non-zero error

mpt_raid_phys_disk_pg0

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_raid_phys_disk_pg0` — returns phys disk page zero

Synopsis

```
int mpt_raid_phys_disk_pg0 (MPT_ADAPTER * ioc, u8
phys_disk_num, RaidPhysDiskPage0_t * phys_disk);
```

Arguments

`ioc`

Pointer to a Adapter Structure

`phys_disk_num`

io unit unique phys disk num generated by the ioc

`phys_disk`

requested payload data returned

Return

0 on success -EFAULT if read of config page header fails or data pointer not NULL -ENOMEM if pci_alloc failed

mpt_raid_phys_disk_get_num_paths

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_raid_phys_disk_get_num_paths` — returns number paths associated to this phys_num

Synopsis

```
int mpt_raid_phys_disk_get_num_paths (MPT_ADAPTER * ioc, u8
phys_disk_num);
```

Arguments

ioc

Pointer to a Adapter Structure

phys_disk_num

io unit unique phys disk num generated by the ioc

Return

returns number paths

mpt_raid_phys_disk_pg1

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_raid_phys_disk_pg1` — returns phys disk page 1

Synopsis

```
int mpt_raid_phys_disk_pg1 (MPT_ADAPTER * ioc, u8
phys_disk_num, RaidPhysDiskPage1_t * phys_disk);
```

Arguments

ioc

Pointer to a Adapter Structure

phys_disk_num

io unit unique phys disk num generated by the ioc

phys_disk

requested payload data returned

Return

0 on success -EFAULT if read of config page header fails or data pointer not NULL -ENOMEM if pci_alloc failed

mpt_findImVolumes

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_findImVolumes` — Identify IDs of hidden disks and RAID Volumes

Synopsis

```
int mpt_findImVolumes (MPT_ADAPTER * ioc);
```

Arguments

`ioc`

Pointer to a Adapter Strucutre

Return

0 on success -EFAULT if read of config page header fails or data pointer not NULL -ENOMEM if pci_alloc failed

mpt_config

LINUX

Name

`mpt_config` — Generic function to issue config message

Synopsis

```
int mpt_config (MPT_ADAPTER * ioc, CONFIGPARMS * pCfg);
```

Arguments

ioc

Pointer to an adapter structure

pCfg

Pointer to a configuration structure. Struct contains action, page address, direction, physical address and pointer to a configuration page header Page header is updated.

Description

Returns 0 for success -EPERM if not allowed due to ISR context -EAGAIN if no msg frames currently available -EFAULT for non-successful reply or no reply (timeout)

mpt_print_ioc_summary

LINUX

Name

`mpt_print_ioc_summary` — Write ASCII summary of IOC to a buffer.

Synopsis

```
void mpt_print_ioc_summary (MPT_ADAPTER * ioc, char * buffer,
int * size, int len, int showlan);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

buffer

Pointer to buffer where IOC summary info should be written

size

Pointer to number of bytes we wrote (set by this routine)

len

Offset at which to start writing in buffer

showlan

Display LAN stuff?

Description

This routine writes (english readable) ASCII text, which represents a summary of IOC information, to a buffer.

mpt_set_taskmgmt_in_progress_flag

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_set_taskmgmt_in_progress_flag` — set flags associated with task managment

Synopsis

```
int mpt_set_taskmgmt_in_progress_flag (MPT_ADAPTER * ioc);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

Description

Returns 0 for SUCCESS or -1 if FAILED.

If -1 is return, then it was not possible to set the flags

mpt_clear_taskmgmt_in_progress_flag

LINUX

Name

`mpt_clear_taskmgmt_in_progress_flag` — clear flags associated with task managment

Synopsis

```
void mpt_clear_taskmgmt_in_progress_flag (MPT_ADAPTER * ioc);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

mpt_halt_firmware

LINUX

Name

`mpt_halt_firmware` — Halts the firmware if it is operational and panic the kernel

Synopsis

```
void mpt_halt_firmware (MPT_ADAPTER * ioc);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

mpt_Soft_Hard_ResetHandler

LINUX

Kernel Hackers ManualApril 2016

Name

mpt_Soft_Hard_ResetHandler — Try less expensive reset

Synopsis

```
int mpt_Soft_Hard_ResetHandler (MPT_ADAPTER * ioc, int
sleepFlag);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

sleepFlag

Indicates if sleep or schedule must be called.

Description

Returns 0 for SUCCESS or -1 if FAILED. Try for softreset first, only if it fails go for expensive HardReset.

mpt_HardResetHandler

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_HardResetHandler` — Generic reset handler

Synopsis

```
int mpt_HardResetHandler (MPT_ADAPTER * ioc, int sleepFlag);
```

Arguments

`ioc`

Pointer to MPT_ADAPTER structure

`sleepFlag`

Indicates if sleep or schedule must be called.

Description

Issues SCSI Task Management call based on input arg values. If TaskMgmt fails, returns associated SCSI request.

Remark

_HardResetHandler can be invoked from an interrupt thread (timer) or a non-interrupt thread. In the former, must not call `schedule`.

Note

A return of -1 is a FATAL error case, as it means a FW reload/initialization failed.

Returns 0 for SUCCESS or -1 if FAILED.

mpt_set_debug_level

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_set_debug_level` — global setting of the mpt_debug_level found via /sys/module/mptbase/parameters/mpt_debug_level

Synopsis

```
int mpt_set_debug_level (const char * val, struct kernel_param
* kp);
```

Arguments

`val`

-- undescribed --

`kp`

Description

Returns

mpt_get_cb_idx

LINUX

Kernel Hackers ManualApril 2016

Name

mpt_get_cb_idx — obtain cb_idx for registered driver

Synopsis

```
u8 mpt_get_cb_idx (MPT_DRIVER_CLASS dclass);
```

Arguments

dclass

class driver enum

Description

Returns cb_idx, or zero means it wasn't found

mpt_is_discovery_complete

LINUX

Name

`mpt_is_discovery_complete` — determine if discovery has completed

Synopsis

```
int mpt_is_discovery_complete (MPT_ADAPTER * ioc);
```

Arguments

ioc

per adatper instance

Description

Returns 1 when discovery completed, else zero.

mpt_remove_dead_ioc_func

LINUX

Name

`mpt_remove_dead_ioc_func` — kthread context to remove dead ioc

Synopsis

```
int mpt_remove_dead_ioc_func (void * arg);
```

Arguments

arg

input argument, used to derive ioc

Description

Return 0 if controller is removed from pci subsystem. Return -1 for other case.

mpt_fault_reset_work

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_fault_reset_work` — work performed on workq after ioc fault

Synopsis

```
void mpt_fault_reset_work (struct work_struct * work);
```

Arguments

work

input argument, used to derive ioc

mpt_interrupt

LINUX

Kernel Hackers ManualApril 2016

Name

mpt_interrupt — MPT adapter (IOC) specific interrupt handler.

Synopsis

```
irqreturn_t mpt_interrupt (int irq, void * bus_id);
```

Arguments

irq

irq number (not used)

bus_id

bus identifier cookie == pointer to MPT_ADAPTER structure

Description

This routine is registered via the request_irq kernel API call, and handles all interrupts generated from a specific MPT adapter (also referred to as a IO Controller or IOC). This routine must clear the interrupt from the adapter and does

so by reading the reply FIFO. Multiple replies may be processed per single call to this routine.

This routine handles register-level access of the adapter but dispatches (calls) a protocol-specific callback routine to handle the protocol-specific details of the MPT request completion.

mptbase_reply

LINUX

Kernel Hackers ManualApril 2016

Name

`mptbase_reply` — MPT base driver's callback routine

Synopsis

```
int mptbase_reply (MPT_ADAPTER * ioc, MPT_FRAME_HDR * req,
MPT_FRAME_HDR * reply);
```

Arguments

`ioc`

Pointer to MPT_ADAPTER structure

`req`

Pointer to original MPT request frame

`reply`

Pointer to MPT reply frame (NULL if TurboReply)

Description

MPT base driver's callback routine; all base driver "internal" request/reply processing is routed here. Currently used for EventNotification and EventAck handling.

Returns 1 indicating original alloc'd request frame ptr should be freed, or 0 if it shouldn't.

mpt_add_sge

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_add_sge` — Place a simple 32 bit SGE at address pAddr.

Synopsis

```
void mpt_add_sge (void * pAddr, u32 flagslength, dma_addr_t
dma_addr);
```

Arguments

`pAddr`

virtual address for SGE

`flagslength`

SGE flags and data transfer length

`dma_addr`

Physical address

Description

This routine places a MPT request frame back on the MPT adapter's FreeQ.

mpt_add_sge_64bit

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_add_sge_64bit` — Place a simple 64 bit SGE at address pAddr.

Synopsis

```
void mpt_add_sge_64bit (void * pAddr, u32 flagslength,
dma_addr_t dma_addr);
```

Arguments

pAddr

virtual address for SGE

flagslength

SGE flags and data transfer length

dma_addr

Physical address

Description

This routine places a MPT request frame back on the MPT adapter's FreeQ.

mpt_add_sge_64bit_1078

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_add_sge_64bit_1078` — Place a simple 64 bit SGE at address pAddr (1078 workaround).

Synopsis

```
void mpt_add_sge_64bit_1078 (void * pAddr, u32 flagslength,
dma_addr_t dma_addr);
```

Arguments

pAddr

virtual address for SGE

flagslength

SGE flags and data transfer length

dma_addr

Physical address

Description

This routine places a MPT request frame back on the MPT adapter's FreeQ.

mpt_add_chain

LINUX

Kernel Hackers ManualApril 2016

Name

mpt_add_chain — Place a 32 bit chain SGE at address pAddr.

Synopsis

```
void mpt_add_chain (void * pAddr, u8 next, u16 length,
dma_addr_t dma_addr);
```

Arguments

pAddr

virtual address for SGE

next

nextChainOffset value (u32's)

length

length of next SGL segment

dma_addr

Physical address

mpt_add_chain_64bit

LINUX

Name

`mpt_add_chain_64bit` — Place a 64 bit chain SGE at address pAddr.

Synopsis

```
void mpt_add_chain_64bit (void * pAddr, u8 next, u16 length,
dma_addr_t dma_addr);
```

Arguments

pAddr

virtual address for SGE

next

nextChainOffset value (u32's)

length

length of next SGL segment

dma_addr

Physical address

mpt_host_page_access_control

LINUX

Name

`mpt_host_page_access_control` — control the IOC's Host Page Buffer access

Synopsis

```
int mpt_host_page_access_control (MPT_ADAPTER * ioc, u8
access_control_value, int sleepFlag);
```

Arguments

`ioc`

Pointer to MPT adapter structure

`access_control_value`

define bits below

`sleepFlag`

Specifies whether the process can sleep

Description

Provides mechanism for the host driver to control the IOC's Host Page Buffer access.

Access Control Value - bits[15:12] 0h Reserved 1h Enable Access { MPI_DB_HPBAC_ENABLE_ACCESS } 2h Disable Access { MPI_DB_HPBAC_DISABLE_ACCESS } 3h Free Buffer { MPI_DB_HPBAC_FREE_BUFFER }

Returns 0 for success, non-zero for failure.

mpt_host_page_alloc

LINUX

Kernel Hackers ManualApril 2016

Name

mpt_host_page_alloc — allocate system memory for the fw

Synopsis

```
int mpt_host_page_alloc (MPT_ADAPTER * ioc, pIOCInit_t
ioc_init);
```

Arguments

ioc

Pointer to pointer to IOC adapter

ioc_init

Pointer to ioc init config page

Description

If we already allocated memory in past, then resend the same pointer. Returns 0 for success, non-zero for failure.

mpt_get_product_name

LINUX

Name

`mpt_get_product_name` — returns product string

Synopsis

```
const char* mpt_get_product_name (u16 vendor, u16 device, u8
revision);
```

Arguments

vendor

pci vendor id

device

pci device id

revision

pci revision id

Description

Returns product string displayed when driver loads, in /proc/mpt/summary and /sysfs/class/scsi_host/host<X>/version_product

convert_to_kb

LINUX

Name

`convert_to_kb` — map in memory mapped io

Synopsis

```
convert_to_kb ( x);
```

Arguments

x

-- undescribed --

mpt_do_ioc_recovery

LINUX

Name

`mpt_do_ioc_recovery` — Initialize or recover MPT adapter.

Synopsis

```
int mpt_do_ioc_recovery (MPT_ADAPTER * ioc, u32 reason, int
sleepFlag);
```

Arguments

ioc

Pointer to MPT adapter structure

reason

Event word / reason

sleepFlag

Use schedule if CAN_SLEEP else use udelay.

Description

This routine performs all the steps necessary to bring the IOC to a OPERATIONAL state.

This routine also pre-fetches the LAN MAC address of a Fibre Channel MPT adapter.

Returns

0 for success -1 if failed to get board READY -2 if READY but IOCFacts Failed -3 if READY but PrimeIOCFifos Failed -4 if READY but IOCInit Failed -5 if failed to enable_device and/or request_selected_regions -6 if failed to upload firmware

mpt_detect_bound_ports

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_detect_bound_ports` — Search for matching PCI bus/dev_function

Synopsis

```
void mpt_detect_bound_ports (MPT_ADAPTER * ioc, struct pci_dev
* pdev);
```

Arguments

ioc

Pointer to MPT adapter structure

pdev

Pointer to (struct pci_dev) structure

Description

Search for PCI bus/dev_function which matches PCI bus/dev_function (+/-1) for newly discovered 929, 929X, 1030 or 1035.

If match on PCI dev_function +/-1 is found, bind the two MPT adapters using alt_ioc pointer fields in their `MPT_ADAPTER` structures.

mpt_adapter_disable

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_adapter_disable` — Disable misbehaving MPT adapter.

Synopsis

```
void mpt_adapter_disable (MPT_ADAPTER * ioc);
```

Arguments

ioc

Pointer to MPT adapter structure

mpt_adapter_dispose

LINUX

Kernel Hackers ManualApril 2016

Name

mpt_adapter_dispose — Free all resources associated with an MPT adapter

Synopsis

```
void mpt_adapter_dispose (MPT_ADAPTER * ioc);
```

Arguments

ioc

Pointer to MPT adapter structure

Description

This routine unregisters h/w resources and frees all alloc'd memory associated with a MPT adapter structure.

MptDisplayIocCapabilities

LINUX

Kernel Hackers ManualApril 2016

Name

`MptDisplayIocCapabilities` — Disply IOC's capabilities.

Synopsis

```
void MptDisplayIocCapabilities (MPT_ADAPTER * ioc);
```

Arguments

ioc

Pointer to MPT adapter structure

MakeIocReady

LINUX

Name

`MakeIocReady` — Get IOC to a READY state, using KickStart if needed.

Synopsis

```
int MakeIocReady (MPT_ADAPTER * ioc, int force, int
sleepFlag);
```

Arguments

`ioc`

Pointer to MPT_ADAPTER structure

`force`

Force hard KickStart of IOC

`sleepFlag`

Specifies whether the process can sleep

Returns

1 - DIAG reset and READY 0 - READY initially OR soft reset and READY -1 - Any failure on KickStart -2 - Msg Unit Reset Failed -3 - IO Unit Reset Failed -4 - IOC owned by a PEER

GetIocFacts

LINUX

Name

`GetIocFacts` — Send IOCFacts request to MPT adapter.

Synopsis

```
int GetIocFacts (MPT_ADAPTER * ioc, int sleepFlag, int
reason);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

sleepFlag

Specifies whether the process can sleep

reason

If recovery, only update facts.

Description

Returns 0 for success, non-zero for failure.

GetPortFacts

LINUX

Name

`GetPortFacts` — Send PortFacts request to MPT adapter.

Synopsis

```
int GetPortFacts (MPT_ADAPTER * ioc, int portnum, int
sleepFlag);
```

Arguments

`ioc`

Pointer to MPT_ADAPTER structure

`portnum`

Port number

`sleepFlag`

Specifies whether the process can sleep

Description

Returns 0 for success, non-zero for failure.

SendIocInit

LINUX

Name

`SendIocInit` — Send IOCInit request to MPT adapter.

Synopsis

```
int SendIocInit (MPT_ADAPTER * ioc, int sleepFlag);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

sleepFlag

Specifies whether the process can sleep

Description

Send IOCInit followed by PortEnable to bring IOC to OPERATIONAL state.

Returns 0 for success, non-zero for failure.

SendPortEnable

LINUX

Name

`SendPortEnable` — Send PortEnable request to MPT adapter port.

Synopsis

```
int SendPortEnable (MPT_ADAPTER * ioc, int portnum, int
sleepFlag);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

portnum

Port number to enable

sleepFlag

Specifies whether the process can sleep

Description

Send PortEnable to bring IOC to OPERATIONAL state.

Returns 0 for success, non-zero for failure.

mpt_do_upload

LINUX

Kernel Hackers ManualApril 2016

Name

mpt_do_upload — Construct and Send FWUpload request to MPT adapter port.

Synopsis

```
int mpt_do_upload (MPT_ADAPTER * ioc, int sleepFlag);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

sleepFlag

Specifies whether the process can sleep

Description

Returns 0 for success, >0 for handshake failure <0 for fw upload failure.

Remark

If bound IOC and a successful FWUpload was performed on the bound IOC, the second image is discarded and memory is free'd. Both channels must upload to prevent IOC from running in degraded mode.

mpt_downloadboot

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_downloadboot` — DownloadBoot code

Synopsis

```
int mpt_downloadboot (MPT_ADAPTER * ioc, MpiFwHeader_t *
pFwHeader, int sleepFlag);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

pFwHeader

Pointer to firmware header info

sleepFlag

Specifies whether the process can sleep

Description

FwDownloadBoot requires Programmed IO access.

Returns 0 for success -1 FW Image size is 0 -2 No valid cached_fw Pointer <0 for fw upload failure.

KickStart

LINUX

Kernel Hackers ManualApril 2016

Name

`KickStart` — Perform hard reset of MPT adapter.

Synopsis

```
int KickStart (MPT_ADAPTER * ioc, int force, int sleepFlag);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

force

Force hard reset

sleepFlag

Specifies whether the process can sleep

Description

This routine places MPT adapter in diagnostic mode via the WriteSequence register, and then performs a hard reset of adapter via the Diagnostic register.

Inputs

sleepflag - CAN_SLEEP (non-interrupt thread) or NO_SLEEP (interrupt thread, use mdelay) force - 1 if doorbell active, board fault state board operational, IOC_RECOVERY or IOC_BRINGUP and there is an alt_ioc. 0 else

Returns

1 - hard reset, READY 0 - no reset due to History bit, READY -1 - no reset due to History bit but not READY OR reset but failed to come READY -2 - no reset, could not enter DIAG mode -3 - reset but bad FW bit

mpt_diag_reset

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_diag_reset` — Perform hard reset of the adapter.

Synopsis

```
int mpt_diag_reset (MPT_ADAPTER * ioc, int ignore, int
sleepFlag);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

ignore

Set if to honor and clear to ignore the reset history bit

sleepFlag

CAN_SLEEP if called in a non-interrupt thread, else set to NO_SLEEP (use mdelay instead)

Description

This routine places the adapter in diagnostic mode via the WriteSequence register and then performs a hard reset of adapter via the Diagnostic register. Adapter should be in ready state upon successful completion.

Returns

1 hard reset successful 0 no reset performed because reset history bit set -2 enabling diagnostic mode failed -3 diagnostic reset failed

SendIocReset

LINUX

Kernel Hackers ManualApril 2016

Name

`SendIocReset` — Send IOCReset request to MPT adapter.

Synopsis

```
int SendIocReset (MPT_ADAPTER * ioc, u8 reset_type, int
sleepFlag);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

reset_type

reset type, expected values are `MPI_FUNCTION_IOC_MESSAGE_UNIT_RESET` or `MPI_FUNCTION_IO_UNIT_RESET`

sleepFlag

Specifies whether the process can sleep

Description

Send IOCReset request to the MPT adapter.

Returns 0 for success, non-zero for failure.

initChainBuffers

LINUX

Kernel Hackers ManualApril 2016

Name

`initChainBuffers` — Allocate memory for and initialize chain buffers

Synopsis

```
int initChainBuffers (MPT_ADAPTER * ioc);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

Description

Allocates memory for and initializes chain buffers, chain buffer control arrays and spinlock.

PrimeIocFifos

LINUX

Kernel Hackers ManualApril 2016

Name

`PrimeIocFifos` — Initialize IOC request and reply FIFOs.

Synopsis

```
int PrimeIocFifos (MPT_ADAPTER * ioc);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

Description

This routine allocates memory for the MPT reply and request frame pools (if necessary), and primes the IOC reply FIFO with reply frames.

Returns 0 for success, non-zero for failure.

mpt_handshake_req_reply_wait

LINUX

Name

`mpt_handshake_req_reply_wait` — Send MPT request to and receive reply from IOC via doorbell handshake method.

Synopsis

```
int mpt_handshake_req_reply_wait (MPT_ADAPTER * ioc, int
reqBytes, u32 * req, int replyBytes, u16 * u16reply, int
maxwait, int sleepFlag);
```

Arguments

`ioc`

Pointer to MPT_ADAPTER structure

`reqBytes`

Size of the request in bytes

`req`

Pointer to MPT request frame

`replyBytes`

Expected size of the reply in bytes

`u16reply`

Pointer to area where reply should be written

`maxwait`

Max wait time for a reply (in seconds)

`sleepFlag`

Specifies whether the process can sleep

NOTES

It is the callers responsibility to byte-swap fields in the request which are greater than 1 byte in size. It is also the callers responsibility to byte-swap response fields which are greater than 1 byte in size.

Returns 0 for success, non-zero for failure.

WaitForDoorbellAck

LINUX

Kernel Hackers ManualApril 2016

Name

`WaitForDoorbellAck` — Wait for IOC doorbell handshake acknowledge

Synopsis

```
int WaitForDoorbellAck (MPT_ADAPTER * ioc, int howlong, int
sleepFlag);
```

Arguments

`ioc`

Pointer to MPT_ADAPTER structure

`howlong`

How long to wait (in seconds)

`sleepFlag`

Specifies whether the process can sleep

Description

This routine waits (up to ~2 seconds max) for IOC doorbell handshake ACKnowledge, indicated by the IOP_DOORBELL_STATUS bit in its IntStatus register being clear.

Returns a negative value on failure, else wait loop count.

WaitForDoorbellInt

LINUX

Kernel Hackers ManualApril 2016

Name

`WaitForDoorbellInt` — Wait for IOC to set its doorbell interrupt bit

Synopsis

```
int WaitForDoorbellInt (MPT_ADAPTER * ioc, int howlong, int
sleepFlag);
```

Arguments

`ioc`

Pointer to MPT_ADAPTER structure

`howlong`

How long to wait (in seconds)

`sleepFlag`

Specifies whether the process can sleep

Description

This routine waits (up to ~2 seconds max) for IOC doorbell interrupt (MPI_HIS_DOORBELL_INTERRUPT) to be set in the IntStatus register.

Returns a negative value on failure, else wait loop count.

WaitForDoorbellReply

LINUX

Kernel Hackers ManualApril 2016

Name

`WaitForDoorbellReply` — Wait for and capture an IOC handshake reply.

Synopsis

```
int WaitForDoorbellReply (MPT_ADAPTER * ioc, int howlong, int
sleepFlag);
```

Arguments

`ioc`

Pointer to MPT_ADAPTER structure

`howlong`

How long to wait (in seconds)

`sleepFlag`

Specifies whether the process can sleep

Description

This routine polls the IOC for a handshake reply, 16 bits at a time. Reply is cached to IOC private area large enough to hold a maximum of 128 bytes of reply data.

Returns a negative value on failure, else size of reply in WORDS.

GetLanConfigPages

LINUX

Kernel Hackers ManualApril 2016

Name

`GetLanConfigPages` — Fetch LANConfig pages.

Synopsis

```
int GetLanConfigPages (MPT_ADAPTER * ioc);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

Return

0 for success -ENOMEM if no memory available -EPERM if not allowed due to ISR context -EAGAIN if no msg frames currently available -EFAULT for non-successful reply or no reply (timeout)

GetIoUnitPage2

LINUX

Kernel Hackers ManualApril 2016

Name

GetIoUnitPage2 — Retrieve BIOS version and boot order information.

Synopsis

```
int GetIoUnitPage2 (MPT_ADAPTER * ioc);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

Returns

0 for success -ENOMEM if no memory available -EPERM if not allowed due to ISR context -EAGAIN if no msg frames currently available -EFAULT for non-successful reply or no reply (timeout)

mpt_GetScsiPortSettings

LINUX

Name

mpt_GetScsiPortSettings — read SCSI Port Page 0 and 2

Synopsis

```
int mpt_GetScsiPortSettings (MPT_ADAPTER * ioc, int portnum);
```

Arguments

ioc

Pointer to a Adapter Strucutre

portnum

IOC port number

Return

-EFAULT if read of config page header fails or if no nvram If read of SCSI Port Page 0 fails, NVRAM = MPT_HOST_NVRAM_INVALID (0xFFFFFFFF)

Adapter settings

async, narrow Return 1 If read of SCSI Port Page 2 fails, Adapter settings valid NVRAM = MPT_HOST_NVRAM_INVALID (0xFFFFFFFF) Return 1 Else Both valid Return 0 CHECK - what type of locking mechanisms should be used????

mpt_readScsiDevicePageHeaders

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_readScsiDevicePageHeaders` — save version and length of SDP1

Synopsis

```
int mpt_readScsiDevicePageHeaders (MPT_ADAPTER * ioc, int portnum);
```

Arguments

ioc

Pointer to a Adapter Strucutre

portnum

IOC port number

Return

-EFAULT if read of config page header fails or 0 if success.

mpt_inactive_raid_list_free

LINUX

Name

`mpt_inactive_raid_list_free` — This clears this link list.

Synopsis

```
void mpt_inactive_raid_list_free (MPT_ADAPTER * ioc);
```

Arguments

`ioc`

pointer to per adapter structure

mpt_inactive_raid_volumes

LINUX

Name

`mpt_inactive_raid_volumes` — sets up link list of phy_disk_nums for devices belonging in an inactive volume

Synopsis

```
void mpt_inactive_raid_volumes (MPT_ADAPTER * ioc, u8 channel,
u8 id);
```

Arguments

ioc

pointer to per adapter structure

channel

volume channel

id

volume target id

mpt_sort_ioc_pg2

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_sort_ioc_pg2` — compare function for sorting volumes in ascending order

Synopsis

```
int mpt_sort_ioc_pg2 (const void * a, const void * b);
```

Arguments

a

ioc_pg2 raid volume page

b

ioc_pg2 raid volume page

Return

0 same, 1 (a is bigger), -1 (b is bigger)

SendEventNotification

LINUX

Kernel Hackers ManualApril 2016

Name

`SendEventNotification` — Send EventNotification (on or off) request to adapter

Synopsis

```
int SendEventNotification (MPT_ADAPTER * ioc, u8 EvSwitch, int sleepFlag);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

EvSwitch

Event switch flags

sleepFlag

Specifies whether the process can sleep

SendEventAck

LINUX

Kernel Hackers ManualApril 2016

Name

SendEventAck — Send EventAck request to MPT adapter.

Synopsis

```
int SendEventAck (MPT_ADAPTER * ioc, EventNotificationReply_t
* evnp);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

evnp

Pointer to original EventNotification request

mpt_ioc_reset

LINUX

Name

`mpt_ioc_reset` — Base cleanup for hard reset

Synopsis

```
int mpt_ioc_reset (MPT_ADAPTER * ioc, int reset_phase);
```

Arguments

`ioc`

Pointer to the adapter structure

`reset_phase`

Indicates pre- or post-reset functionality

Remark

Frees resources with internally generated commands.

procmpt_create

LINUX

Kernel Hackers ManualApril 2016

Name

`procmpt_create` — Create `MPT_PROCFS_MPTBASEDIR` entries.

Synopsis

```
int procmpt_create ( void);
```

Arguments

void

no arguments

Description

Returns 0 for success, non-zero for failure.

procmpt_destroy

LINUX

Kernel Hackers ManualApril 2016

Name

procmpt_destroy — Tear down MPT_PROCFS_MPTBASEDIR entries.

Synopsis

```
void procmpt_destroy ( void);
```

Arguments

void

no arguments

Description

Returns 0 for success, non-zero for failure.

procmpt_summary_read

LINUX

Kernel Hackers ManualApril 2016

Name

`procmpt_summary_read` — Handle read request of a summary file

Synopsis

```
int procmpt_summary_read (char * buf, char ** start, off_t
offset, int request, int * eof, void * data);
```

Arguments

buf

Pointer to area to write information

start

Pointer to start pointer

offset

Offset to start writing

request

Amount of read data requested

eof

Pointer to EOF integer

data

Pointer

Description

Handles read request from /proc/mpt/summary or /proc/mpt/iocN/summary.
Returns number of characters written to process performing the read.

procmpt_version_read

LINUX

Kernel Hackers ManualApril 2016

Name

`procmpt_version_read` — Handle read request from /proc/mpt/version.

Synopsis

```
int procmpt_version_read (char * buf, char ** start, off_t
offset, int request, int * eof, void * data);
```

Arguments

buf

Pointer to area to write information

start

Pointer to start pointer

offset

Offset to start writing

request

Amount of read data requested

eof

Pointer to EOF integer

data

Pointer

Description

Returns number of characters written to process performing the read.

procmpt_iocinfo_read

LINUX

Kernel Hackers ManualApril 2016

Name

`procmpt_iocinfo_read` — Handle read request from /proc/mpt/iocN/info.

Synopsis

```
int procmpt_iocinfo_read (char * buf, char ** start, off_t
offset, int request, int * eof, void * data);
```

Arguments

buf

Pointer to area to write information

start

Pointer to start pointer

offset

Offset to start writing

request

Amount of read data requested

eof

Pointer to EOF integer

data

Pointer

Description

Returns number of characters written to process performing the read.

mpt_SoftResetHandler

LINUX

Name

`mpt_SoftResetHandler` — Issues a less expensive reset

Synopsis

```
int mpt_SoftResetHandler (MPT_ADAPTER * ioc, int sleepFlag);
```

Arguments

`ioc`

Pointer to MPT_ADAPTER structure

`sleepFlag`

Indicates if sleep or schedule must be called.

Description

Returns 0 for SUCCESS or -1 if FAILED.

Message Unit Reset - instructs the IOC to reset the Reply Post and Free FIFO's. All the Message Frames on Reply Free FIFO are discarded. All posted buffers are freed, and event notification is turned off. IOC doesn't reply to any outstanding request. This will transfer IOC to READY state.

ProcessEventNotification

LINUX

Name

`ProcessEventNotification` — Route EventNotificationReply to all event handlers

Synopsis

```
int ProcessEventNotification (MPT_ADAPTER * ioc,
EventNotificationReply_t * pEventReply, int * evHandlers);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

pEventReply

Pointer to EventNotification reply frame

evHandlers

Pointer to integer, number of event handlers

Description

Routes a received EventNotificationReply to all currently registered event handlers. Returns sum of event handlers return values.

mpt_fc_log_info

LINUX

Name

`mpt_fc_log_info` — Log information returned from Fibre Channel IOC.

Synopsis

```
void mpt_fc_log_info (MPT_ADAPTER * ioc, u32 log_info);
```

Arguments

`ioc`

Pointer to MPT_ADAPTER structure

`log_info`

U32 LogInfo reply word from the IOC

Description

Refer to lsi/mpi_log_fc.h.

mpt_spi_log_info

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_spi_log_info` — Log information returned from SCSI Parallel IOC.

Synopsis

```
void mpt_spi_log_info (MPT_ADAPTER * ioc, u32 log_info);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

log_info

U32 LogInfo word from the IOC

Description

Refer to lsi/sp_log.h.

mpt_sas_log_info

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_sas_log_info` — Log information returned from SAS IOC.

Synopsis

```
void mpt_sas_log_info (MPT_ADAPTER * ioc, u32 log_info, u8
cb_idx);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

log_info

U32 LogInfo reply word from the IOC

cb_idx

-- undescribed --

Description

Refer to lsi/mpi_log_sas.h.

mpt_iocstatus_info_config

LINUX

Kernel Hackers ManualApril 2016

Name

`mpt_iocstatus_info_config` — IOCSTATUS information for config pages

Synopsis

```
void mpt_iocstatus_info_config (MPT_ADAPTER * ioc, u32
ioc_status, MPT_FRAME_HDR * mf);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

ioc_status

U32 IOCStatus word from IOC

mf

Pointer to MPT request frame

Description

Refer to lsi/mpi.h.

mpt_iocstatus_info

LINUX

Kernel Hackers ManualApril 2016

Name

mpt_iocstatus_info — IOCSTATUS information returned from IOC.

Synopsis

```
void mpt_iocstatus_info (MPT_ADAPTER * ioc, u32 ioc_status,
MPT_FRAME_HDR * mf);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

ioc_status

U32 IOCStatus word from IOC

mf

Pointer to MPT request frame

Description

Refer to lsi/mpi.h.

fusion_init

LINUX

Kernel Hackers ManualApril 2016

Name

`fusion_init` — Fusion MPT base driver initialization routine.

Synopsis

```
int fusion_init ( void);
```

Arguments

void

no arguments

Description

Returns 0 for success, non-zero for failure.

fusion_exit

LINUX

Kernel Hackers ManualApril 2016

Name

`fusion_exit` — Perform driver unload cleanup.

Synopsis

```
void __exit fusion_exit ( void);
```

Arguments

void

no arguments

Description

This routine frees all resources associated with each MPT adapter and removes all `MPT_PROCFS_MPTBASEDIR` entries.

mptscsih_info

LINUX

Kernel Hackers ManualApril 2016

Name

`mptscsih_info` — Return information about MPT adapter

Synopsis

```
const char * mptscsih_info (struct Scsi_Host * SChost);
```

Arguments

`SChost`

Pointer to Scsi_Host structure

Description

(linux scsi_host_template.info routine)

Returns pointer to buffer where information was written.

mptscsih_proc_info

LINUX

Kernel Hackers ManualApril 2016

Name

`mptscsih_proc_info` — Return information about MPT adapter

Synopsis

```
int mptscsih_proc_info (struct Scsi_Host * host, char *
buffer, char ** start, off_t offset, int length, int func);
```

Arguments

host

scsi host struct

buffer

if write, user data; if read, buffer for user

start

returns the buffer address

offset

if write, 0; if read, the current offset into the buffer from the previous read.

length

if write, return length;

func

write = 1; read = 0

Description

(linux scsi_host_template.info routine)

mptscsih_qcmd

LINUX

Kernel Hackers ManualApril 2016

Name

`mptscsih_qcmd` — Primary Fusion MPT SCSI initiator IO start routine.

Synopsis

```
int mptscsih_qcmd (struct scsi_cmnd * SCpnt);
```

Arguments

SCpnt

Pointer to scsi_cmnd structure

Description

(linux scsi_host_template.queuecommand routine) This is the primary SCSI IO start routine. Create a MPI SCSIIORequest from a linux scsi_cmnd request and send it to the IOC.

Returns 0. (rtn value discarded by linux scsi mid-layer)

mptscsih_IssueTaskMgmt

LINUX

Kernel Hackers ManualApril 2016

Name

`mptscsih_IssueTaskMgmt` — Generic send Task Management function.

Synopsis

```
int mptscsih_IssueTaskMgmt (MPT_SCSI_HOST * hd, u8 type, u8
channel, u8 id, int lun, int ctx2abort, ulong timeout);
```

Arguments

hd

Pointer to MPT_SCSI_HOST structure

type

Task Management type

channel

channel number for task management

id

Logical Target ID for reset (if appropriate)

lun

Logical Unit for reset (if appropriate)

ctx2abort

Context for the task to be aborted (if appropriate)

`timeout`

timeout for task management control

Remark

_HardResetHandler can be invoked from an interrupt thread (timer) or a non-interrupt thread. In the former, must not call `schedule`.

Not all fields are meaningfull for all task types.

Returns 0 for SUCCESS, or FAILED.

mptscsih_abort

LINUX

Kernel Hackers ManualApril 2016

Name

`mptscsih_abort` — Abort linux scsi_cmnd routine, new_eh variant

Synopsis

```
int mptscsih_abort (struct scsi_cmnd * SCpnt);
```

Arguments

`SCpnt`

Pointer to scsi_cmnd structure, IO to be aborted

Description

(linux scsi_host_template.eh_abort_handler routine)

Returns SUCCESS or FAILED.

mptscsih_dev_reset

LINUX

Kernel Hackers ManualApril 2016

Name

`mptscsih_dev_reset` — Perform a SCSI TARGET_RESET! new_eh variant

Synopsis

```
int mptscsih_dev_reset (struct scsi_cmnd * SCpnt);
```

Arguments

SCpnt

Pointer to scsi_cmnd structure, IO which reset is due to

Description

(linux scsi_host_template.eh_dev_reset_handler routine)

Returns SUCCESS or FAILED.

mptscsih_bus_reset

LINUX

Kernel Hackers ManualApril 2016

Name

`mptscsih_bus_reset` — Perform a SCSI BUS_RESET! new_eh variant

Synopsis

```
int mptscsih_bus_reset (struct scsi_cmnd * SCpnt);
```

Arguments

SCpnt

Pointer to scsi_cmnd structure, IO which reset is due to

Description

(linux scsi_host_template.eh_bus_reset_handler routine)

Returns SUCCESS or FAILED.

mptscsih_host_reset

LINUX

Name

`mptscsih_host_reset` — Perform a SCSI host adapter RESET (new_eh variant)

Synopsis

```
int mptscsih_host_reset (struct scsi_cmnd * SCpnt);
```

Arguments

`SCpnt`

Pointer to scsi_cmnd structure, IO which reset is due to

Description

(linux scsi_host_template.eh_host_reset_handler routine)

Returns SUCCESS or FAILED.

mptscsih_taskmgmt_complete

LINUX

Name

`mptscsih_taskmgmt_complete` — Registered with Fusion MPT base driver

Synopsis

```
int mptscsih_taskmgmt_complete (MPT_ADAPTER * ioc,
MPT_FRAME_HDR * mf, MPT_FRAME_HDR * mr);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

mf

Pointer to SCSI task mgmt request frame

mr

Pointer to SCSI task mgmt reply frame

Description

This routine is called from mptbase.c::`mpt_interrupt` at the completion of any SCSI task management request. This routine is registered with the MPT (base) driver at driver load/init time via the `mpt_register` API call.

Returns 1 indicating alloc'd request frame ptr should be freed.

mptscsih_is_phys_disk

LINUX

Kernel Hackers ManualApril 2016

Name

`mptscsih_is_phys_disk` —

Synopsis

```
int mptscsih_is_phys_disk (MPT_ADAPTER * ioc, u8 channel, u8
id);
```

Arguments

ioc

-- undescribed --

channel

-- undescribed --

id

-- undescribed --

mptscsih_get_scsi_lookup

LINUX

Kernel Hackers ManualApril 2016

Name

mptscsih_get_scsi_lookup —

Synopsis

```
struct scsi_cmnd * mptscsih_get_scsi_lookup (MPT_ADAPTER *
ioc, int i);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

i

index into the array

Description

Returns the scsi_cmd pointer

Description

Returns the scsi_cmd pointer

mptscsih_do_cmd

LINUX

Kernel Hackers ManualApril 2016

Name

`mptscsih_do_cmd` — Do internal command.

Synopsis

```
int mptscsih_do_cmd (MPT_SCSI_HOST * hd, INTERNAL_CMD * io);
```

Arguments

hd

MPT_SCSI_HOST pointer

io

INTERNAL_CMD pointer.

Description

Issue the specified internally generated command and do command specific cleanup. For bus scan / DV only.

NOTES

If command is Inquiry and status is good, initialize a target structure, save the data

Remark

Single threaded access only.

Return

< 0 if an illegal command or no resources

0 if good

> 0 if command complete but some type of completion error.

mptscsih_info_scsiio

LINUX

Name

`mptscsih_info_scsiio` — debug print info on reply frame

Synopsis

```
void mptscsih_info_scsiio (MPT_ADAPTER * ioc, struct scsi_cmnd
* sc, SCSIIOReply_t * pScsiReply);
```

Arguments

`ioc`

Pointer to MPT_ADAPTER structure

`sc`

original scsi cmnd pointer

`pScsiReply`

Pointer to MPT reply frame

Description

MPT_DEBUG_REPLY needs to be enabled to obtain this info

Refer to lsi/mpi.h.

_scsih_setup_eedp

LINUX

Name

`_scsih_setup_eedp` — setup MPI request for EEDP transfer

Synopsis

```
int _scsih_setup_eedp (MPT_ADAPTER * ioc, struct scsi_cmnd *
scmd, SCSIIO32Request_t * mpi_request);
```

Arguments

ioc

-- undescribed --

scmd

pointer to scsi command object

mpi_request

pointer to the SCSI_IO reqest message frame

Description

Supporting protection 1 only.

Returns nothing

_scsih_read_capacity_16

LINUX

Name

`_scsih_read_capacity_16` — send READ_CAPACITY_16 to target

Synopsis

```
int _scsih_read_capacity_16 (MPT_SCSI_HOST * hd, int id, int
channel, u32 lun, void * data, u32 length);
```

Arguments

hd

-- undescribed --

id

-- undescribed --

channel

-- undescribed --

lun

-- undescribed --

data

-- undescribed --

length

-- undescribed --

Description

mptscsih_getclear_scsi_lookup

LINUX

Kernel Hackers ManualApril 2016

Name

mptscsih_getclear_scsi_lookup —

Synopsis

```
struct scsi_cmnd * mptscsih_getclear_scsi_lookup (MPT_ADAPTER
* ioc, int i);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

i

index into the array

Description

Returns the scsi_cmd pointer

Description

Returns the scsi_cmd pointer

mptscsih_set_scsi_lookup

LINUX

Kernel Hackers ManualApril 2016

Name

mptscsih_set_scsi_lookup —

Synopsis

```
void mptscsih_set_scsi_lookup (MPT_ADAPTER * ioc, int i,
struct scsi_cmnd * scmd);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

i

index into the array

scmd

scsi_cmnd pointer

Description

writes a scmd entry into the ScsiLookup[] array list

SCPNT_TO_LOOKUP_IDX

LINUX

Kernel Hackers ManualApril 2016

Name

SCPNT_TO_LOOKUP_IDX — searches for a given scmd in the ScsiLookup[] array list

Synopsis

```
int SCPNT_TO_LOOKUP_IDX (MPT_ADAPTER * ioc, struct scsi_cmnd * sc);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

sc

scsi_cmnd pointer

mptscsih_get_completion_code

LINUX

Kernel Hackers ManualApril 2016

Name

mptscsih_get_completion_code —

Synopsis

```
int mptscsih_get_completion_code (MPT_ADAPTER * ioc,
MPT_FRAME_HDR * req, MPT_FRAME_HDR * reply);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

req

-- undescribed --

reply

-- undescribed --

mptscsih_synchronize_cache

LINUX

Kernel Hackers ManualApril 2016

Name

mptscsih_synchronize_cache — Send SYNCHRONIZE_CACHE to all disks.

Synopsis

```
void mptscsih_synchronize_cache (struct scsi_device * sdev,
MPT_SCSI_HOST * hd, VirtDevice * vdevice);
```

Arguments

sdev

-- undescribed --

hd

Pointer to a SCSI HOST structure

vdevice

virtual target device

Description

Uses the ISR, but with special processing. MUST be single-threaded.

mptscsih_device_sas_address_show

LINUX

Kernel Hackers ManualApril 2016

Name

`mptscsih_device_sas_address_show` — sas address *cdev* - pointer to embedded class device *buf* - the buffer returned

Synopsis

```
ssize_t mptscsih_device_sas_address_show (struct device * dev,
struct device_attribute * attr, char * buf);
```

Arguments

dev

-- undescribed --

attr

-- undescribed --

buf

-- undescribed --

Description

This is the sas address for the target

A sysfs 'read-only' shost attribute.

mptscsih_device_handle_show

LINUX

Kernel Hackers ManualApril 2016

Name

mptscsih_device_handle_show — device handle *cdev* - pointer to embedded class device *buf* - the buffer returned

Synopsis

```
ssize_t mptscsih_device_handle_show (struct device * dev,
struct device_attribute * attr, char * buf);
```

Arguments

dev

-- undescribed --

attr

-- undescribed --

buf

-- undescribed --

Description

This is the firmware assigned device handle

A sysfs 'read-only' shost attribute.

mptscsih_device_fw_id_show

LINUX

Kernel Hackers ManualApril 2016

Name

mptscsih_device_fw_id_show — device handle *cdev* - pointer to embedded class device *buf* - the buffer returned

Synopsis

```
ssize_t mptscsih_device_fw_id_show (struct device * dev,
struct device_attribute * attr, char * buf);
```

Arguments

dev

-- undescribed --

attr

-- undescribed --

buf

-- undescribed --

Description

This is the firmware assigned id.

A sysfs 'read-only' shost attribute.

mptctl_syscall_down

LINUX

Kernel Hackers ManualApril 2016

Name

`mptctl_syscall_down` — Down the MPT adapter syscall semaphore.

Synopsis

```
int mptctl_syscall_down (MPT_ADAPTER * ioc, int nonblock);
```

Arguments

ioc

Pointer to MPT adapter

nonblock

boolean, non-zero if O_NONBLOCK is set

Description

All of the ioctl commands can potentially sleep, which is illegal with a spinlock held, thus we perform mutual exclusion here.

Returns negative errno on error, or zero for success.

mptspi_setTargetNegoParms

LINUX

Kernel Hackers ManualApril 2016

Name

`mptspi_setTargetNegoParms` — Update the target negotiation parameters

Synopsis

```
void mptspi_setTargetNegoParms (MPT_SCSI_HOST * hd, VirtTarget
* target, struct scsi_device * sdev);
```

Arguments

hd

Pointer to a SCSI Host Structure

target

per target private data

sdev

SCSI device

Description

Update the target negotiation parameters based on the the Inquiry data, adapter capabilities, and NVRAM settings.

mptspi_writeIOCPage4

LINUX

Kernel Hackers ManualApril 2016

Name

`mptspi_writeIOCPage4` — write IOC Page 4

Synopsis

```
int mptspi_writeIOCPage4 (MPT_SCSI_HOST * hd, u8 channel, u8
id);
```

Arguments

hd

Pointer to a SCSI Host Structure

channel

channel number

id

write IOC Page4 for this ID & Bus

Return

-EAGAIN if unable to obtain a Message Frame or 0 if success.

Remark

We do not wait for a return, write pages sequentially.

mptspi_initTarget

LINUX

Kernel Hackers ManualApril 2016

Name

`mptspi_initTarget` — Target, LUN alloc/free functionality.

Synopsis

```
void mptspi_initTarget (MPT_SCSI_HOST * hd, VirtTarget *
vtarget, struct scsi_device * sdev);
```

Arguments

hd

Pointer to MPT_SCSI_HOST structure

vtarget

per target private data

sdev

SCSI device

NOTE

It's only SAFE to call this routine if data points to sane & valid STANDARD INQUIRY data!

Allocate and initialize memory for this target. Save inquiry data.

mptspi_is_raid

LINUX

Kernel Hackers ManualApril 2016

Name

`mptspi_is_raid` — Determines whether target is belonging to volume

Synopsis

```
int mptspi_is_raid (struct _MPT_SCSI_HOST * hd, u32 id);
```

Arguments

hd

Pointer to a SCSI HOST structure

id

target device id

Return

non-zero = true zero = false

mptspi_print_write_nego

LINUX

Kernel Hackers ManualApril 2016

Name

`mptspi_print_write_nego` — negotiation parameters debug info that is being sent

Synopsis

```
void mptspi_print_write_nego (struct _MPT_SCSI_HOST * hd,
struct scsi_target * starget, u32 ii);
```

Arguments

hd

Pointer to a SCSI HOST structure

starget

SCSI target

ii

negotiation parameters

mptspi_print_read_nego

LINUX

Kernel Hackers ManualApril 2016

Name

mptspi_print_read_nego — negotiation parameters debug info that is being read

Synopsis

```
void mptspi_print_read_nego (struct _MPT_SCSI_HOST * hd,
struct scsi_target * starget, u32 ii);
```

Arguments

hd

Pointer to a SCSI HOST structure

starget

SCSI target

ii

negotiation parameters

mptspi_dv_renegotiate_work

LINUX

Kernel Hackers ManualApril 2016

Name

mptspi_dv_renegotiate_work —

Synopsis

```
void mptspi_dv_renegotiate_work (struct work_struct * work);
```

Arguments

work

-- undescribed --

mptspi_ioc_reset

LINUX

Kernel Hackers ManualApril 2016

Name

mptspi_ioc_reset —

Synopsis

```
int mptspi_ioc_reset (MPT_ADAPTER * ioc, int reset_phase);
```

Arguments

ioc

-- undescribed --

reset_phase

-- undescribed --

mptspi_resume

LINUX

Kernel Hackers ManualApril 2016

Name

mptspi_resume —

Synopsis

```
int mptspi_resume (struct pci_dev * pdev);
```

Arguments

pdev

-- undescribed --

mptspi_probe

LINUX

Kernel Hackers ManualApril 2016

Name

`mptspi_probe` — Installs scsi devices per bus.

Synopsis

```
int mptspi_probe (struct pci_dev * pdev, const struct
pci_device_id * id);
```

Arguments

pdev

Pointer to pci_dev structure

id

-- undescribed --

Description

Returns 0 for success, non-zero for failure.

mptspi_init

LINUX

Kernel Hackers ManualApril 2016

Name

`mptspi_init` — Register MPT adapter(s) as SCSI host(s) with SCSI mid-layer.

Synopsis

```
int mptspi_init ( void);
```

Arguments

void

no arguments

Description

Returns 0 for success, non-zero for failure.

mptspi_exit

LINUX

Name

`mptspi_exit` — Unregisters MPT adapter(s)

Synopsis

```
void __exit mptspi_exit ( void);
```

Arguments

void

no arguments

mptfc_set_sdev_queue_depth

LINUX

Name

`mptfc_set_sdev_queue_depth` — global setting of the mpt_sdev_queue_depth found via /sys/module/mptfc/parameters/mpt_sdev_queue_depth

Synopsis

```
int mptfc_set_sdev_queue_depth (const char * val, struct
kernel_param * kp);
```

Arguments

val

-- undescribed --

kp

Description

Returns

mptfc_init

LINUX

Kernel Hackers ManualApril 2016

Name

`mptfc_init` — Register MPT adapter(s) as SCSI host(s) with SCSI mid-layer.

Synopsis

```
int mptfc_init ( void);
```

Arguments

void

no arguments

Description

Returns 0 for success, non-zero for failure.

mptfc_remove

LINUX

Kernel Hackers ManualApril 2016

Name

`mptfc_remove` — Remove fc infrastructure for devices

Synopsis

```
void __devexit mptfc_remove (struct pci_dev * pdev);
```

Arguments

pdev

Pointer to pci_dev structure

mptfc_exit

LINUX

Kernel Hackers ManualApril 2016

Name

`mptfc_exit` — Unregisters MPT adapter(s)

Synopsis

```
void __exit mptfc_exit ( void);
```

Arguments

`void`

no arguments

lan_reply

LINUX

Kernel Hackers ManualApril 2016

Name

`lan_reply` — Handle all data sent from the hardware.

Synopsis

```
int lan_reply (MPT_ADAPTER * ioc, MPT_FRAME_HDR * mf,
MPT_FRAME_HDR * reply);
```

Arguments

ioc

Pointer to MPT_ADAPTER structure

mf

Pointer to original MPT request frame (NULL if TurboReply)

reply

Pointer to MPT reply frame

Description

Returns 1 indicating original alloc'd request frame ptr should be freed, or 0 if it shouldn't.

4.2. I2O message devices

i2o_driver_notify_controller_add

LINUX

Name

`i2o_driver_notify_controller_add` — Send notification of added controller

Synopsis

```
void i2o_driver_notify_controller_add (struct i2o_driver *
drv, struct i2o_controller * c);
```

Arguments

drv

I2O driver

c

I2O controller

Description

Send notification of added controller to a single registered driver.

i2o_driver_notify_controller_remove

LINUX

Name

`i2o_driver_notify_controller_remove` — Send notification of removed controller

Synopsis

```
void i2o_driver_notify_controller_remove (struct i2o_driver *
drv, struct i2o_controller * c);
```

Arguments

drv

I2O driver

c

I2O controller

Description

Send notification of removed controller to a single registered driver.

i2o_driver_notify_device_add

LINUX

Name

`i2o_driver_notify_device_add` — Send notification of added device

Synopsis

```
void i2o_driver_notify_device_add (struct i2o_driver * drv,
struct i2o_device * i2o_dev);
```

Arguments

drv

I2O driver

i2o_dev

the added i2o_device

Description

Send notification of added device to a single registered driver.

i2o_driver_notify_device_remove

LINUX

Name

`i2o_driver_notify_device_remove` — Send notification of removed

device

Synopsis

```
void i2o_driver_notify_device_remove (struct i2o_driver * drv,
struct i2o_device * i2o_dev);
```

Arguments

drv

I2O driver

i2o_dev

the added i2o_device

Description

Send notification of removed device to a single registered driver.

i2o_msg_out_to_virt

LINUX

Kernel Hackers ManualApril 2016

Name

i2o_msg_out_to_virt — Turn an I2O message to a virtual address

Synopsis

```
struct i2o_message * i2o_msg_out_to_virt (struct
i2o_controller * c, u32 m);
```

Arguments

c

controller

m

message engine value

Description

Turn a receive message from an I2O controller bus address into a Linux virtual address. The shared page frame is a linear block so we simply have to shift the offset. This function does not work for sender side messages as they are ioremap objects provided by the I2O controller.

i2o_msg_in_to_virt

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_msg_in_to_virt` — Turn an I2O message to a virtual address

Synopsis

```
struct i2o_message __iomem * i2o_msg_in_to_virt (struct
i2o_controller * c, u32 m);
```

Arguments

c

controller

m

message engine value

Description

Turn a send message from an I2O controller bus address into a Linux virtual address. The shared page frame is a linear block so we simply have to shift the offset. This function does not work for receive side messages as they are kmalloc objects in a different pool.

i2o_msg_get

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_msg_get` — obtain an I2O message from the IOP

Synopsis

```
struct i2o_message * i2o_msg_get (struct i2o_controller * c);
```

Arguments

c

I2O controller

Description

This function tries to get a message frame. If no message frame is available do not wait until one is available (see also i2o_msg_get_wait). The returned pointer to the message frame is not in I/O memory, it is allocated from a mempool. But because a MFA is allocated from the controller too it is guaranteed that `i2o_msg_post` will never fail.

On a success a pointer to the message frame is returned. If the message queue is empty -EBUSY is returned and if no memory is available -ENOMEM is returned.

i2o_msg_post

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_msg_post` — Post I2O message to I2O controller

Synopsis

```
void i2o_msg_post (struct i2o_controller * c, struct
i2o_message * msg);
```

Arguments

c

I2O controller to which the message should be send

msg

message returned by `i2o_msg_get`

Description

Post the message to the I2O controller and return immediately.

i2o_msg_post_wait

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_msg_post_wait` — Post and wait a message and wait until return

Synopsis

```
int i2o_msg_post_wait (struct i2o_controller * c, struct
i2o_message * msg, unsigned long timeout);
```

Arguments

c

controller

msg

message to post

timeout

time in seconds to wait

Description

This API allows an OSM to post a message and then be told whether or not the system received a successful reply. If the message times out then the value '-ETIMEDOUT' is returned.

Returns 0 on success or negative error code on failure.

i2o_msg_nop_mfa

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_msg_nop_mfa` — Returns a fetched MFA back to the controller

Synopsis

```
void i2o_msg_nop_mfa (struct i2o_controller * c, u32 mfa);
```

Arguments

c

I2O controller from which the MFA was fetched

mfa

MFA which should be returned

Description

This function must be used for preserved messages, because `i2o_msg_nop` also returns the allocated memory back to the msg_pool mempool.

i2o_msg_nop

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_msg_nop` — Returns a message which is not used

Synopsis

```
void i2o_msg_nop (struct i2o_controller * c, struct
i2o_message * msg);
```

Arguments

c

I2O controller from which the message was created

msg

message which should be returned

Description

If you fetch a message via i2o_msg_get, and can't use it, you must return the message with this function. Otherwise the MFA is lost as well as the allocated memory from the mempool.

i2o_flush_reply

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_flush_reply` — Flush reply from I2O controller

Synopsis

```
void i2o_flush_reply (struct i2o_controller * c, u32 m);
```

Arguments

c

I2O controller

m

the message identifier

Description

The I2O controller must be informed that the reply message is not needed anymore. If you forget to flush the reply, the message frame can't be used by the controller anymore and is therefore lost.

i2o_iop_free

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_iop_free` — Free the i2o_controller struct

Synopsis

```
void i2o_iop_free (struct i2o_controller * c);
```

Arguments

c

I2O controller to free

i2o_msg_get_wait

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_msg_get_wait` — obtain an I2O message from the IOP

Synopsis

```
struct i2o_message * i2o_msg_get_wait (struct i2o_controller *
c, int wait);
```

Arguments

c

I2O controller

wait

how long to wait until timeout

Description

This function waits up to wait seconds for a message slot to be available.

On a success the message is returned and the pointer to the message is set in msg. The returned message is the physical page frame offset address from the read port (see the i2o spec). If no message is available returns I2O_QUEUE_EMPTY and msg is leaved untouched.

i2o_cntxt_list_add

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_cntxt_list_add` — Append a pointer to context list and return a id

Synopsis

```
u32 i2o_cntxt_list_add (struct i2o_controller * c, void *
ptr);
```

Arguments

c

controller to which the context list belong

ptr

pointer to add to the context list

Description

Because the context field in I2O is only 32-bit large, on 64-bit the pointer is to large to fit in the context field. The i2o_cntxt_list functions therefore map pointers to context fields.

Returns context id > 0 on success or 0 on failure.

i2o_cntxt_list_remove

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_cntxt_list_remove` — Remove a pointer from the context list

Synopsis

```
u32 i2o_cntxt_list_remove (struct i2o_controller * c, void * ptr);
```

Arguments

c

controller to which the context list belong

ptr

pointer which should be removed from the context list

Description

Removes a previously added pointer from the context list and returns the matching context id.

Returns context id on success or 0 on failure.

i2o_cntxt_list_get

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_cntxt_list_get` — Get a pointer from the context list and remove it

Synopsis

```
void * i2o_cntxt_list_get (struct i2o_controller * c, u32 context);
```

Arguments

c

controller to which the context list belong

context

context id to which the pointer belong

Description

Returns pointer to the matching context id on success or NULL on failure.

i2o_cntxt_list_get_ptr

LINUX

Name

`i2o_cntxt_list_get_ptr` — Get a context id from the context list

Synopsis

```
u32 i2o_cntxt_list_get_ptr (struct i2o_controller * c, void *
ptr);
```

Arguments

c

controller to which the context list belong

ptr

pointer to which the context id should be fetched

Description

Returns context id which matches to the pointer on success or 0 on failure.

i2o_find_iop

LINUX

Name

`i2o_find_iop` — Find an I2O controller by id

Synopsis

```
struct i2o_controller * i2o_find_iop (int unit);
```

Arguments

unit

unit number of the I2O controller to search for

Description

Lookup the I2O controller on the controller list.

Returns pointer to the I2O controller on success or NULL if not found.

i2o_iop_find_device

LINUX

Kernel Hackers ManualApril 2016

Name

i2o_iop_find_device — Find a I2O device on an I2O controller

Synopsis

```
struct i2o_device * i2o_iop_find_device (struct i2o_controller
* c, u16 tid);
```

Arguments

c

I2O controller where the I2O device hangs on

tid

TID of the I2O device to search for

Description

Searches the devices of the I2O controller for a device with TID tid and returns it.

Returns a pointer to the I2O device if found, otherwise NULL.

i2o_status_get

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_status_get` — Get the status block from the I2O controller

Synopsis

```
int i2o_status_get (struct i2o_controller * c);
```

Arguments

c

I2O controller

Description

Issue a status query on the controller. This updates the attached status block. The status block could then be accessed through c->status_block.

Returns 0 on success or negative error code on failure.

i2o_event_register

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_event_register` — Turn on/off event notification for a I2O device

Synopsis

```
int i2o_event_register (struct i2o_device * dev, struct
i2o_driver * drv, int tcntxt, u32 evt_mask);
```

Arguments

dev

I2O device which should receive the event registration request

drv

driver which want to get notified

tcntxt

transaction context to use with this notifier

evt_mask

mask of events

Description

Create and posts an event registration message to the task. No reply is waited for, or expected. If you do not want further notifications, call the i2o_event_register again with a evt_mask of 0.

Returns 0 on success or negative error code on failure.

i2o_iop_quiesce

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_iop_quiesce` — quiesce controller

Synopsis

```
int i2o_iop_quiesce (struct i2o_controller * c);
```

Arguments

c

controller

Description

Quiesce an IOP. Causes IOP to make external operation quiescent (i2o 'READY' state). Internal operation of the IOP continues normally.

Returns 0 on success or negative error code on failure.

i2o_iop_enable

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_iop_enable` — move controller from ready to OPERATIONAL

Synopsis

```
int i2o_iop_enable (struct i2o_controller * c);
```

Arguments

c

I2O controller

Description

Enable IOP. This allows the IOP to resume external operations and reverses the effect of a quiesce. Returns zero or an error code if an error occurs.

i2o_iop_quiesce_all

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_iop_quiesce_all` — Quiesce all I2O controllers on the system

Synopsis

```
void i2o_iop_quiesce_all ( void);
```

Arguments

void

no arguments

Description

Quiesce all I2O controllers which are connected to the system.

i2o_iop_enable_all

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_iop_enable_all` — Enables all controllers on the system

Synopsis

```
void i2o_iop_enable_all ( void);
```

Arguments

void

no arguments

Description

Enables all I2O controllers which are connected to the system.

i2o_iop_clear

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_iop_clear` — Bring I2O controller into HOLD state

Synopsis

```
int i2o_iop_clear (struct i2o_controller * c);
```

Arguments

c

controller

Description

Clear an IOP to HOLD state, ie. terminate external operations, clear all input queues and prepare for a system restart. IOP's internal operation continues normally and the outbound queue is alive. The IOP is not expected to rebuild its LCT.

Returns 0 on success or negative error code on failure.

i2o_iop_init_outbound_queue

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_iop_init_outbound_queue` — setup the outbound message queue

Synopsis

```
int i2o_iop_init_outbound_queue (struct i2o_controller * c);
```

Arguments

c

I2O controller

Description

Clear and (re)initialize IOP's outbound queue and post the message frames to the IOP.

Returns 0 on success or negative error code on failure.

i2o_iop_reset

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_iop_reset` — reset an I2O controller

Synopsis

```
int i2o_iop_reset (struct i2o_controller * c);
```

Arguments

c

controller to reset

Description

Reset the IOP into INIT state and wait until IOP gets into RESET state. Terminate all external operations, clear IOP's inbound and outbound queues, terminate all DDMs, and reload the IOP's operating environment and all local DDMs. The IOP rebuilds its LCT.

i2o_iop_activate

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_iop_activate` — Bring controller up to HOLD

Synopsis

```
int i2o_iop_activate (struct i2o_controller * c);
```

Arguments

c

controller

Description

This function brings an I2O controller into HOLD state. The adapter is reset if necessary and then the queues and resource table are read.

Returns 0 on success or negative error code on failure.

i2o_iop_systab_set

LINUX

Name

`i2o_iop_systab_set` — Set the I2O System Table of the specified IOP

Synopsis

```
int i2o_iop_systab_set (struct i2o_controller * c);
```

Arguments

c

I2O controller to which the system table should be send

Description

Before the systab could be set `i2o_systab_build` must be called.

Returns 0 on success or negative error code on failure.

i2o_iop_online

LINUX

Name

`i2o_iop_online` — Bring a controller online into OPERATIONAL state.

Synopsis

```
int i2o_iop_online (struct i2o_controller * c);
```

Arguments

c

I2O controller

Description

Send the system table and enable the I2O controller.

Returns 0 on success or negative error code on failure.

i2o_iop_remove

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_iop_remove` — Remove the I2O controller from the I2O core

Synopsis

```
void i2o_iop_remove (struct i2o_controller * c);
```

Arguments

c

I2O controller

Description

Remove the I2O controller from the I2O core. If devices are attached to the controller remove these also and finally reset the controller.

i2o_systab_build

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_systab_build` — Build system table

Synopsis

```
int i2o_systab_build ( void);
```

Arguments

void

no arguments

Description

The system table contains information about all the IOPs in the system (duh) and is used by the Executives on the IOPs to establish peer2peer connections. We're not supporting peer2peer at the moment, but this will be needed down the road for things like lan2lan forwarding.

Returns 0 on success or negative error code on failure.

i2o_parse_hrt

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_parse_hrt` — Parse the hardware resource table.

Synopsis

```
int i2o_parse_hrt (struct i2o_controller * c);
```

Arguments

c

I2O controller

Description

We don't do anything with it except dumping it (in debug mode).

Returns 0.

i2o_iop_release

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_iop_release` — release the memory for a I2O controller

Synopsis

```
void i2o_iop_release (struct device * dev);
```

Arguments

dev

I2O controller which should be released

Description

Release the allocated memory. This function is called if refcount of device reaches 0 automatically.

i2o_iop_alloc

LINUX

Name

`i2o_iop_alloc` — Allocate and initialize a i2o_controller struct

Synopsis

```
struct i2o_controller * i2o_iop_alloc ( void);
```

Arguments

`void`

no arguments

Description

Allocate the necessary memory for a i2o_controller struct and initialize the lists and message mempool.

Returns a pointer to the I2O controller or a negative error code on failure.

i2o_iop_add

LINUX

Name

`i2o_iop_add` — Initialize the I2O controller and add him to the I2O core

Synopsis

```
int i2o_iop_add (struct i2o_controller * c);
```

Arguments

c

controller

Description

Initialize the I2O controller and if no error occurs add him to the I2O core.

Returns 0 on success or negative error code on failure.

i2o_iop_init

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_iop_init` — I2O main initialization function

Synopsis

```
int i2o_iop_init ( void);
```

Arguments

void

no arguments

Description

Initialize the I2O drivers (OSM) functions, register the Executive OSM, initialize the I2O PCI part and finally initialize I2O device stuff.

Returns 0 on success or negative error code on failure.

i2o_iop_exit

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_iop_exit` — I2O main exit function

Synopsis

```
void __exit i2o_iop_exit ( void);
```

Arguments

void

no arguments

Description

Removes I2O controllers from PCI subsystem and shut down OSMs.

i2o_config_init

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_config_init` — Configuration OSM initialization function

Synopsis

```
int i2o_config_init ( void);
```

Arguments

`void`

no arguments

Description

Registers Configuration OSM in the I2O core and if old ioctl's are compiled in initialize them.

Returns 0 on success or negative error code on failure.

i2o_config_exit

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_config_exit` — Configuration OSM exit function

Synopsis

```
void i2o_config_exit ( void);
```

Arguments

`void`

no arguments

Description

If old ioctl's are compiled in exit remove them and unregisters Configuration OSM from I2O core.

i2o_msg_post_wait_mem

LINUX

Name

`i2o_msg_post_wait_mem` — Post and wait a message with DMA buffers

Synopsis

```
int i2o_msg_post_wait_mem (struct i2o_controller * c, struct
i2o_message * msg, unsigned long timeout, struct i2o_dma *
dma);
```

Arguments

c

controller

msg

message to post

timeout

time in seconds to wait

dma

i2o_dma struct of the DMA buffer to free on failure

Description

This API allows an OSM to post a message and then be told whether or not the system received a successful reply. If the message times out then the value '-ETIMEDOUT' is returned. This is a special case. In this situation the message may (should) complete at an indefinite time in the future. When it completes it will use the memory buffer attached to the request. If -ETIMEDOUT is returned then the memory buffer must not be freed. Instead the event completion will free them for you. In all other cases the buffer are your problem.

Returns 0 on success, negative error code on timeout or positive error code from reply.

i2o_exec_lct_get

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_exec_lct_get` — Get the IOP's Logical Configuration Table

Synopsis

```
int i2o_exec_lct_get (struct i2o_controller * c);
```

Arguments

c

I2O controller from which the LCT should be fetched

Description

Send a LCT NOTIFY request to the controller, and wait I2O_TIMEOUT_LCT_GET seconds until arrival of response. If the LCT is to large, retry it.

Returns 0 on success or negative error code on failure.

i2o_exec_wait_alloc

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_exec_wait_alloc` — Allocate a i2o_exec_wait struct an initialize it

Synopsis

```
struct i2o_exec_wait * i2o_exec_wait_alloc ( void);
```

Arguments

void

no arguments

Description

Allocate the i2o_exec_wait struct and initialize the wait.

Returns i2o_exec_wait pointer on success or negative error code on failure.

i2o_exec_wait_free

LINUX

Name

`i2o_exec_wait_free` — Free an i2o_exec_wait struct

Synopsis

```
void i2o_exec_wait_free (struct i2o_exec_wait * wait);
```

Arguments

wait

I2O wait data which should be cleaned up

i2o_msg_post_wait_complete

LINUX

Name

`i2o_msg_post_wait_complete` — Reply to a i2o_msg_post request from IOP

Synopsis

```
int i2o_msg_post_wait_complete (struct i2o_controller * c, u32
m, struct i2o_message * msg, u32 context);
```

Arguments

c

I2O controller which answers

m

message id

msg

pointer to the I2O reply message

context

transaction context of request

Description

This function is called in interrupt context only. If the reply reached before the timeout, the i2o_exec_wait struct is filled with the message and the task will be waked up. The task is now responsible for returning the message m back to the controller! If the message reaches us after the timeout clean up the i2o_exec_wait struct (including allocated DMA buffer).

Return 0 on success and if the message m should not be given back to the I2O controller, or >0 on success and if the message should be given back afterwords. Returns negative error code on failure. In this case the message must also be given back to the controller.

i2o_exec_show_vendor_id

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_exec_show_vendor_id` — Displays Vendor ID of controller

Synopsis

```
ssize_t i2o_exec_show_vendor_id (struct device * d, struct
device_attribute * attr, char * buf);
```

Arguments

d

device of which the Vendor ID should be displayed

attr

device_attribute to display

buf

buffer into which the Vendor ID should be printed

Description

Returns number of bytes printed into buffer.

i2o_exec_show_product_id

LINUX

Kernel Hackers ManualApril 2016

Name

i2o_exec_show_product_id — Displays Product ID of controller

Synopsis

```
ssize_t i2o_exec_show_product_id (struct device * d, struct
device_attribute * attr, char * buf);
```

Arguments

d

device of which the Product ID should be displayed

attr

device_attribute to display

buf

buffer into which the Product ID should be printed

Description

Returns number of bytes printed into buffer.

i2o_exec_probe

LINUX

Kernel Hackers ManualApril 2016

Name

i2o_exec_probe — Called if a new I2O device (executive class) appears

Synopsis

```
int i2o_exec_probe (struct device * dev);
```

Arguments

dev

I2O device which should be probed

Description

Registers event notification for every event from Executive device. The return is always 0, because we want all devices of class Executive.

Returns 0 on success.

i2o_exec_remove

LINUX

Kernel Hackers ManualApril 2016

Name

i2o_exec_remove — Called on I2O device removal

Synopsis

```
int i2o_exec_remove (struct device * dev);
```

Arguments

dev

I2O device which was removed

Description

Unregisters event notification from Executive I2O device.

Returns 0 on success.

i2o_exec_lct_notify

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_exec_lct_notify` — Send a asynchronus LCT NOTIFY request

Synopsis

```
int i2o_exec_lct_notify (struct i2o_controller * c, u32
change_ind);
```

Arguments

c

I2O controller to which the request should be send

change_ind

change indicator

Description

This function sends a LCT NOTIFY request to the I2O controller with the change indicator change_ind. If the change_ind == 0 the controller replies immediately after the request. If change_ind > 0 the reply is send after change indicator of the LCT is > change_ind.

i2o_exec_lct_modified

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_exec_lct_modified` — Called on LCT NOTIFY reply

Synopsis

```
void i2o_exec_lct_modified (struct work_struct * _work);
```

Arguments

_work

work struct for a specific controller

Description

This function handles asynchronus LCT NOTIFY replies. It parses the new LCT and if the buffer for the LCT was to small sends a LCT NOTIFY again, otherwise send LCT NOTIFY to get informed on next LCT change.

i2o_exec_reply

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_exec_reply` — I2O Executive reply handler

Synopsis

```
int i2o_exec_reply (struct i2o_controller * c, u32 m, struct
i2o_message * msg);
```

Arguments

c

I2O controller from which the reply comes

m

message id

msg

pointer to the I2O reply message

Description

This function is always called from interrupt context. If a POST WAIT reply was received, pass it to the complete function. If a LCT NOTIFY reply was received, a new event is created to handle the update.

Returns 0 on success and if the reply should not be flushed or > 0 on success and if the reply should be flushed. Returns negative error code on failure and if the reply should be flushed.

i2o_exec_event

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_exec_event` — Event handling function

Synopsis

```
void i2o_exec_event (struct work_struct * work);
```

Arguments

`work`

Work item in occurring event

Description

Handles events send by the Executive device. At the moment does not do anything useful.

i2o_exec_init

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_exec_init` — Registers the Exec OSM

Synopsis

```
int i2o_exec_init ( void);
```

Arguments

void

no arguments

Description

Registers the Exec OSM in the I2O core.

Returns 0 on success or negative error code on failure.

i2o_exec_exit

LINUX

Name

`i2o_exec_exit` — Removes the Exec OSM

Synopsis

```
void i2o_exec_exit ( void);
```

Arguments

`void`

no arguments

Description

Unregisters the Exec OSM from the I2O core.

i2o_bus_scan

LINUX

Name

`i2o_bus_scan` — Scan the bus for new devices

Synopsis

```
int i2o_bus_scan (struct i2o_device * dev);
```

Arguments

dev

I2O device of the bus, which should be scanned

Description

Scans the bus dev for new / removed devices. After the scan a new LCT will be fetched automatically.

Returns 0 on success or negative error code on failure.

i2o_bus_store_scan

LINUX

Kernel Hackers ManualApril 2016

Name

i2o_bus_store_scan — Scan the I2O Bus Adapter

Synopsis

```
ssize_t i2o_bus_store_scan (struct device * d, struct
device_attribute * attr, const char * buf, size_t count);
```

Arguments

d

device which should be scanned

attr

device_attribute

buf

output buffer

count

buffer size

Description

Returns count.

i2o_bus_probe

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_bus_probe` — verify if dev is a I2O Bus Adapter device and install it

Synopsis

```
int i2o_bus_probe (struct device * dev);
```

Arguments

dev

device to verify if it is a I2O Bus Adapter device

Description

Because we want all Bus Adapters always return 0. Except when we fail. Then we are sad.

Returns 0, except when we fail to excel.

i2o_bus_remove

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_bus_remove` — remove the I2O Bus Adapter device from the system again

Synopsis

```
int i2o_bus_remove (struct device * dev);
```

Arguments

dev

I2O Bus Adapter device which should be removed

Description

Always returns 0.

i2o_bus_init

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_bus_init` — Bus Adapter OSM initialization function

Synopsis

```
int i2o_bus_init ( void);
```

Arguments

void

no arguments

Description

Only register the Bus Adapter OSM in the I2O core.

Returns 0 on success or negative error code on failure.

i2o_bus_exit

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_bus_exit` — Bus Adapter OSM exit function

Synopsis

```
void __exit i2o_bus_exit ( void);
```

Arguments

void

no arguments

Description

Unregisters Bus Adapter OSM from I2O core.

i2o_device_claim

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_device_claim` — claim a device for use by an OSM

Synopsis

```
int i2o_device_claim (struct i2o_device * dev);
```

Arguments

dev

I2O device to claim

Description

Do the leg work to assign a device to a given OSM. If the claim succeeds, the owner is the primary. If the attempt fails a negative errno code is returned. On success zero is returned.

i2o_device_claim_release

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_device_claim_release` — release a device that the OSM is using

Synopsis

```
int i2o_device_claim_release (struct i2o_device * dev);
```

Arguments

dev

device to release

Description

Drop a claim by an OSM on a given I2O device.

AC - some devices seem to want to refuse an unclaim until they have finished internal processing. It makes sense since you don't want a new device to go reconfiguring the entire system until you are done. Thus we are prepared to wait briefly.

Returns 0 on success or negative error code on failure.

i2o_device_issue_claim

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_device_issue_claim` — claim or release a device

Synopsis

```
int i2o_device_issue_claim (struct i2o_device * dev, u32 cmd,
u32 type);
```

Arguments

dev

I2O device to claim or release

cmd

claim or release command

type

type of claim

Description

Issue I2O UTIL_CLAIM or UTIL_RELEASE messages. The message to be sent is set by cmd. dev is the I2O device which should be claim or released and the type is the claim type (see the I2O spec).

Returs 0 on success or negative error code on failure.

i2o_device_release

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_device_release` — release the memory for a I2O device

Synopsis

```
void i2o_device_release (struct device * dev);
```

Arguments

dev

I2O device which should be released

Description

Release the allocated memory. This function is called if refcount of device reaches 0 automatically.

i2o_device_show_class_id

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_device_show_class_id` — Displays class id of I2O device

Synopsis

```
ssize_t i2o_device_show_class_id (struct device * dev, struct
device_attribute * attr, char * buf);
```

Arguments

dev

device of which the class id should be displayed

attr

pointer to device attribute

buf

buffer into which the class id should be printed

Description

Returns the number of bytes which are printed into the buffer.

i2o_device_show_tid

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_device_show_tid` — Displays TID of I2O device

Synopsis

```
ssize_t i2o_device_show_tid (struct device * dev, struct
device_attribute * attr, char * buf);
```

Arguments

dev

device of which the TID should be displayed

attr

pointer to device attribute

buf

buffer into which the TID should be printed

Description

Returns the number of bytes which are printed into the buffer.

i2o_device_alloc

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_device_alloc` — Allocate a I2O device and initialize it

Synopsis

```
struct i2o_device * i2o_device_alloc ( void);
```

Arguments

void

no arguments

Description

Allocate the memory for a I2O device and initialize locks and lists

Returns the allocated I2O device or a negative error code if the device could not be allocated.

i2o_device_add

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_device_add` — allocate a new I2O device and add it to the IOP

Synopsis

```
int i2o_device_add (struct i2o_controller * c, i2o_lct_entry *
entry);
```

Arguments

c

I2O controller that the device is on

entry

LCT entry of the I2O device

Description

Allocate a new I2O device and initialize it with the LCT entry. The device is appended to the device list of the controller.

Returns zero on success, or a -ve errno.

i2o_device_remove

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_device_remove` — remove an I2O device from the I2O core

Synopsis

```
void i2o_device_remove (struct i2o_device * i2o_dev);
```

Arguments

`i2o_dev`

I2O device which should be released

Description

Is used on I2O controller removal or LCT modification, when the device is removed from the system. Note that the device could still hang around until the refcount reaches 0.

i2o_device_parse_lct

LINUX

Name

`i2o_device_parse_lct` — Parse a previously fetched LCT and create devices

Synopsis

```
int i2o_device_parse_lct (struct i2o_controller * c);
```

Arguments

c

I2O controller from which the LCT should be parsed.

Description

The Logical Configuration Table tells us what we can talk to on the board. For every entry we create an I2O device, which is registered in the I2O core.

Returns 0 on success or negative error code on failure.

i2o_bus_match

LINUX

Name

`i2o_bus_match` — Tell if I2O device class id matches the class ids of the I2O driver (OSM)

Synopsis

```
int i2o_bus_match (struct device * dev, struct device_driver *
drv);
```

Arguments

dev

device which should be verified

drv

the driver to match against

Description

Used by the bus to check if the driver wants to handle the device.

Returns 1 if the class ids of the driver match the class id of the device, otherwise 0.

i2o_driver_dispatch

LINUX

Kernel Hackers ManualApril 2016

Name

i2o_driver_dispatch — dispatch an I2O reply message

Synopsis

```
int i2o_driver_dispatch (struct i2o_controller * c, u32 m);
```

Arguments

c

I2O controller of the message

m

I2O message number

Description

The reply is delivered to the driver from which the original message was. This function is only called from interrupt context.

Returns 0 on success and the message should not be flushed. Returns > 0 on success and if the message should be flushed afterwords. Returns negative error code on failure (the message will be flushed too).

i2o_driver_init

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_driver_init` — initialize I2O drivers (OSMs)

Synopsis

```
int i2o_driver_init ( void);
```

Arguments

void

no arguments

Description

Registers the I2O bus and allocate memory for the array of OSMs.

Returns 0 on success or negative error code on failure.

i2o_driver_exit

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_driver_exit` — clean up I2O drivers (OSMs)

Synopsis

```
void i2o_driver_exit ( void);
```

Arguments

void

no arguments

Description

Unregisters the I2O bus and frees driver array.

i2o_pci_free

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_pci_free` — Frees the DMA memory for the I2O controller

Synopsis

```
void i2o_pci_free (struct i2o_controller * c);
```

Arguments

c

I2O controller to free

Description

Remove all allocated DMA memory and unmap memory IO regions. If MTRR is enabled, also remove it again.

i2o_pci_alloc

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_pci_alloc` — Allocate DMA memory, map IO memory for I2O controller

Synopsis

```
int i2o_pci_alloc (struct i2o_controller * c);
```

Arguments

c

I2O controller

Description

Allocate DMA memory for a PCI (or in theory AGP) I2O controller. All IO mappings are also done here. If MTRR is enabled, also do add memory regions here.

Returns 0 on success or negative error code on failure.

i2o_pci_interrupt

LINUX

Name

`i2o_pci_interrupt` — Interrupt handler for I2O controller

Synopsis

```
irqreturn_t i2o_pci_interrupt (int irq, void * dev_id);
```

Arguments

irq

interrupt line

dev_id

pointer to the I2O controller

Description

Handle an interrupt from a PCI based I2O controller. This turns out to be rather simple. We keep the controller pointer in the cookie.

i2o_pci_irq_enable

LINUX

Name

`i2o_pci_irq_enable` — Allocate interrupt for I2O controller

Synopsis

```
int i2o_pci_irq_enable (struct i2o_controller * c);
```

Arguments

c

i2o_controller that the request is for

Description

Allocate an interrupt for the I2O controller, and activate interrupts on the I2O controller.

Returns 0 on success or negative error code on failure.

i2o_pci_irq_disable

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_pci_irq_disable` — Free interrupt for I2O controller

Synopsis

```
void i2o_pci_irq_disable (struct i2o_controller * c);
```

Arguments

c

I2O controller

Description

Disable interrupts in I2O controller and then free interrupt.

i2o_pci_probe

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_pci_probe` — Probe the PCI device for an I2O controller

Synopsis

```
int i2o_pci_probe (struct pci_dev * pdev, const struct
pci_device_id * id);
```

Arguments

pdev

PCI device to test

id

id which matched with the PCI device id table

Description

Probe the PCI device for any device which is a memory of the Intelligent, I2O class or an Adaptec Zero Channel Controller. We attempt to set up each such device and register it with the core.

Returns 0 on success or negative error code on failure.

i2o_pci_remove

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_pci_remove` — Removes a I2O controller from the system

Synopsis

```
void __devexit i2o_pci_remove (struct pci_dev * pdev);
```

Arguments

pdev

I2O controller which should be removed

Description

Reset the I2O controller, disable interrupts and remove all allocated resources.

i2o_pci_init

LINUX

Kernel Hackers ManualApril 2016

Name

i2o_pci_init — registers I2O PCI driver in PCI subsystem

Synopsis

```
int i2o_pci_init ( void);
```

Arguments

void

no arguments

Description

Returns > 0 on success or negative error code on failure.

i2o_pci_exit

LINUX

Kernel Hackers ManualApril 2016

Name

i2o_pci_exit — unregisters I2O PCI driver from PCI subsystem

Synopsis

```
void __exit i2o_pci_exit ( void);
```

Arguments

void

no arguments

i2o_block_device_free

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_block_device_free` — free the memory of the I2O Block device

Synopsis

```
void i2o_block_device_free (struct i2o_block_device * dev);
```

Arguments

dev

I2O Block device, which should be cleaned up

Description

Frees the request queue, gendisk and the i2o_block_device structure.

i2o_block_remove

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_block_remove` — remove the I2O Block device from the system again

Synopsis

```
int i2o_block_remove (struct device * dev);
```

Arguments

dev

I2O Block device which should be removed

Description

Remove gendisk from system and free all allocated memory.

Always returns 0.

i2o_block_device_flush

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_block_device_flush` — Flush all dirty data of I2O device dev

Synopsis

```
int i2o_block_device_flush (struct i2o_device * dev);
```

Arguments

dev

I2O device which should be flushed

Description

Flushes all dirty data on device dev.

Returns 0 on success or negative error code on failure.

i2o_block_device_mount

LINUX

Name

`i2o_block_device_mount` — Mount (load) the media of device dev

Synopsis

```
int i2o_block_device_mount (struct i2o_device * dev, u32
media_id);
```

Arguments

dev

I2O device which should receive the mount request

media_id

Media Identifier

Description

Load a media into drive. Identifier should be set to -1, because the spec does not support any other value.

Returns 0 on success or negative error code on failure.

i2o_block_device_lock

LINUX

Name

`i2o_block_device_lock` — Locks the media of device dev

Synopsis

```
int i2o_block_device_lock (struct i2o_device * dev, u32
media_id);
```

Arguments

dev

I2O device which should receive the lock request

media_id

Media Identifier

Description

Lock media of device dev to prevent removal. The media identifier should be set to -1, because the spec does not support any other value.

Returns 0 on success or negative error code on failure.

i2o_block_device_unlock

LINUX

Name

`i2o_block_device_unlock` — Unlocks the media of device dev

Synopsis

```
int i2o_block_device_unlock (struct i2o_device * dev, u32
media_id);
```

Arguments

dev

I2O device which should receive the unlocked request

media_id

Media Identifier

Description

Unlocks the media in device dev. The media identifier should be set to -1, because the spec does not support any other value.

Returns 0 on success or negative error code on failure.

i2o_block_device_power

LINUX

Name

`i2o_block_device_power` — Power management for device dev

Synopsis

```
int i2o_block_device_power (struct i2o_block_device * dev, u8
op);
```

Arguments

dev

I2O device which should receive the power management request

op

Operation to send

Description

Send a power management request to the device dev.

Returns 0 on success or negative error code on failure.

i2o_block_request_alloc

LINUX

Name

`i2o_block_request_alloc` — Allocate an I2O block request struct

Synopsis

```
struct i2o_block_request * i2o_block_request_alloc ( void);
```

Arguments

`void`

no arguments

Description

Allocates an I2O block request struct and initialize the list.

Returns a i2o_block_request pointer on success or negative error code on failure.

i2o_block_request_free

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_block_request_free` — Frees a I2O block request

Synopsis

```
void i2o_block_request_free (struct i2o_block_request * ireq);
```

Arguments

ireq

I2O block request which should be freed

Description

Frees the allocated memory (give it back to the request mempool).

i2o_block_sglist_alloc

LINUX

Kernel Hackers ManualApril 2016

Name

i2o_block_sglist_alloc — Allocate the SG list and map it

Synopsis

```
int i2o_block_sglist_alloc (struct i2o_controller * c, struct
i2o_block_request * ireq, u32 ** mptr);
```

Arguments

c

I2O controller to which the request belongs

ireq

I2O block request

mptr

message body pointer

Description

Builds the SG list and map it to be accessible by the controller.

Returns 0 on failure or 1 on success.

i2o_block_sglist_free

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_block_sglist_free` — Frees the SG list

Synopsis

```
void i2o_block_sglist_free (struct i2o_block_request * ireq);
```

Arguments

ireq

I2O block request from which the SG should be freed

Description

Frees the SG list from the I2O block request.

i2o_block_prep_req_fn

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_block_prep_req_fn` — Allocates I2O block device specific struct

Synopsis

```
int i2o_block_prep_req_fn (struct request_queue * q, struct
request * req);
```

Arguments

q

request queue for the request

req

the request to prepare

Description

Allocate the necessary i2o_block_request struct and connect it to the request. This is needed that we not lose the SG list later on.

Returns BLKPREP_OK on success or BLKPREP_DEFER on failure.

i2o_block_delayed_request_fn

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_block_delayed_request_fn` — delayed request queue function

Synopsis

```
void i2o_block_delayed_request_fn (struct work_struct * work);
```

Arguments

work

the delayed request with the queue to start

Description

If the request queue is stopped for a disk, and there is no open request, a new event is created, which calls this function to start the queue after I2O_BLOCK_REQUEST_TIME. Otherwise the queue will never be started again.

i2o_block_end_request

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_block_end_request` — Post-processing of completed commands

Synopsis

```
void i2o_block_end_request (struct request * req, int error,
int nr_bytes);
```

Arguments

req

request which should be completed

error

0 for success, < 0 for error

nr_bytes

number of bytes to complete

Description

Mark the request as complete. The lock must not be held when entering.

i2o_block_reply

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_block_reply` — Block OSM reply handler.

Synopsis

```
int i2o_block_reply (struct i2o_controller * c, u32 m, struct
i2o_message * msg);
```

Arguments

c

I2O controller from which the message arrives

m

message id of reply

msg

the actual I2O message reply

Description

This function gets all the message replies.

i2o_block_open

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_block_open` — Open the block device

Synopsis

```
int i2o_block_open (struct block_device * bdev, fmode_t mode);
```

Arguments

bdev

block device being opened

mode

file open mode

Description

Power up the device, mount and lock the media. This function is called, if the block device is opened for access.

Returns 0 on success or negative error code on failure.

i2o_block_release

LINUX

Name

`i2o_block_release` — Release the I2O block device

Synopsis

```
int i2o_block_release (struct gendisk * disk, fmode_t mode);
```

Arguments

`disk`

gendisk device being released

`mode`

file open mode

Description

Unlock and unmount the media, and power down the device. Gets called if the block device is closed.

Returns 0 on success or negative error code on failure.

i2o_block_ioctl

LINUX

Name

`i2o_block_ioctl` — Issue device specific ioctl calls.

Synopsis

```
int i2o_block_ioctl (struct block_device * bdev, fmode_t mode,
unsigned int cmd, unsigned long arg);
```

Arguments

bdev

block device being opened

mode

file open mode

cmd

ioctl command

arg

arg

Description

Handles ioctl request for the block device.

Return 0 on success or negative error on failure.

i2o_block_check_events

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_block_check_events` — Have we seen a media change?

Synopsis

```
unsigned int i2o_block_check_events (struct gendisk * disk,
unsigned int clearing);
```

Arguments

disk

gendisk which should be verified

clearing

events being cleared

Description

Verifies if the media has changed.

Returns 1 if the media was changed or 0 otherwise.

i2o_block_transfer

LINUX

Name

`i2o_block_transfer` — Transfer a request to/from the I2O controller

Synopsis

```
int i2o_block_transfer (struct request * req);
```

Arguments

req

the request which should be transferred

Description

This function converts the request into a I2O message. The necessary DMA buffers are allocated and after everything is setup post the message to the I2O controller. No cleanup is done by this function. It is done on the interrupt side when the reply arrives.

Return 0 on success or negative error code on failure.

i2o_block_request_fn

LINUX

Name

`i2o_block_request_fn` — request queue handling function

Synopsis

```
void i2o_block_request_fn (struct request_queue * q);
```

Arguments

q

request queue from which the request could be fetched

Description

Takes the next request from the queue, transfers it and if no error occurs dequeue it from the queue. On arrival of the reply the message will be processed further. If an error occurs requeue the request.

i2o_block_device_alloc

LINUX

Kernel Hackers ManualApril 2016

Name

i2o_block_device_alloc — Allocate memory for a I2O Block device

Synopsis

```
struct i2o_block_device * i2o_block_device_alloc ( void);
```

Arguments

void

no arguments

Description

Allocate memory for the i2o_block_device struct, gendisk and request queue and initialize them as far as no additional information is needed.

Returns a pointer to the allocated I2O Block device on success or a negative error code on failure.

i2o_block_probe

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_block_probe` — verify if dev is a I2O Block device and install it

Synopsis

```
int i2o_block_probe (struct device * dev);
```

Arguments

dev

device to verify if it is a I2O Block device

Description

We only verify if the user_tid of the device is 0xfff and then install the device. Otherwise it is used by some other device (e. g. RAID).

Returns 0 on success or negative error code on failure.

i2o_block_init

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_block_init` — Block OSM initialization function

Synopsis

```
int i2o_block_init ( void);
```

Arguments

void

no arguments

Description

Allocate the slab and mempool for request structs, registers i2o_block block device and finally register the Block OSM in the I2O core.

Returns 0 on success or negative error code on failure.

i2o_block_exit

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_block_exit` — Block OSM exit function

Synopsis

```
void __exit i2o_block_exit ( void);
```

Arguments

void

no arguments

Description

Unregisters Block OSM from I2O core, unregisters i2o_block block device and frees the mempool and slab.

i2o_scsi_get_host

LINUX

Name

`i2o_scsi_get_host` — Get an I2O SCSI host

Synopsis

```
struct i2o_scsi_host * i2o_scsi_get_host (struct
i2o_controller * c);
```

Arguments

c

I2O controller to for which to get the SCSI host

Description

If the I2O controller already exists as SCSI host, the SCSI host is returned, otherwise the I2O controller is added to the SCSI core.

Returns pointer to the I2O SCSI host on success or NULL on failure.

i2o_scsi_remove

LINUX

Name

`i2o_scsi_remove` — Remove I2O device from SCSI core

Synopsis

```
int i2o_scsi_remove (struct device * dev);
```

Arguments

dev

device which should be removed

Description

Removes the I2O device from the SCSI core again.

Returns 0 on success.

i2o_scsi_probe

LINUX

Kernel Hackers ManualApril 2016

Name

i2o_scsi_probe — verify if dev is a I2O SCSI device and install it

Synopsis

```
int i2o_scsi_probe (struct device * dev);
```

Arguments

dev

device to verify if it is a I2O SCSI device

Description

Retrieve channel, id and lun for I2O device. If everything goes well register the I2O device as SCSI device on the I2O SCSI controller.

Returns 0 on success or negative error code on failure.

i2o_scsi_reply

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_scsi_reply` — SCSI OSM message reply handler

Synopsis

```
int i2o_scsi_reply (struct i2o_controller * c, u32 m, struct
i2o_message * msg);
```

Arguments

c

controller issuing the reply

m

message id for flushing

msg

the message from the controller

Description

Process reply messages (interrupts in normal scsi controller think). We can get a variety of messages to process. The normal path is scsi command completions. We must also deal with IOP failures, the reply to a bus reset and the reply to a LUN query.

Returns 0 on success and if the reply should not be flushed or > 0 on success and if the reply should be flushed. Returns negative error code on failure and if the reply should be flushed.

i2o_scsi_notify_device_add

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_scsi_notify_device_add` — Retrieve notifications of added devices

Synopsis

```
void i2o_scsi_notify_device_add (struct i2o_device * i2o_dev);
```

Arguments

i2o_dev

the I2O device which was added

Description

If a I2O device is added we catch the notification, because I2O classes other than SCSI peripheral will not be received through `i2o_scsi_probe`.

i2o_scsi_notify_device_remove

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_scsi_notify_device_remove` — Retrieve notifications of removed devices

Synopsis

```
void i2o_scsi_notify_device_remove (struct i2o_device *
i2o_dev);
```

Arguments

i2o_dev

the I2O device which was removed

Description

If a I2O device is removed, we catch the notification to remove the corresponding SCSI device.

i2o_scsi_notify_controller_add

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_scsi_notify_controller_add` — Retrieve notifications of added controllers

Synopsis

```
void i2o_scsi_notify_controller_add (struct i2o_controller *
c);
```

Arguments

c

the controller which was added

Description

If a I2O controller is added, we catch the notification to add a corresponding Scsi_Host.

i2o_scsi_notify_controller_remove

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_scsi_notify_controller_remove` — Retrieve notifications of removed controllers

Synopsis

```
void i2o_scsi_notify_controller_remove (struct i2o_controller
* c);
```

Arguments

c

the controller which was removed

Description

If a I2O controller is removed, we catch the notification to remove the corresponding Scsi_Host.

i2o_scsi_queuecommand_lck

LINUX

Name

`i2o_scsi_queuecommand_lck` — queue a SCSI command

Synopsis

```
int i2o_scsi_queuecommand_lck (struct scsi_cmnd * SCpnt, void
(*done) (struct scsi_cmnd *));
```

Arguments

SCpnt

scsi command pointer

done

callback for completion

Description

Issue a scsi command asynchronously. Return 0 on success or 1 if we hit an error (normally message queue congestion). The only minor complication here is that I2O deals with the device addressing so we have to map the bus/dev/lun back to an I2O handle as well as faking absent devices ourself.

Locks

takes the controller lock on error path only

i2o_scsi_abort

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_scsi_abort` — abort a running command

Synopsis

```
int i2o_scsi_abort (struct scsi_cmnd * SCpnt);
```

Arguments

`SCpnt`

command to abort

Description

Ask the I2O controller to abort a command. This is an asynchrnous process and our callback handler will see the command complete with an aborted message if it succeeds.

Returns 0 if the command is successfully aborted or negative error code on failure.

i2o_scsi_bios_param

LINUX

Name

`i2o_scsi_bios_param` — Invent disk geometry

Synopsis

```
int i2o_scsi_bios_param (struct scsi_device * sdev, struct
block_device * dev, sector_t capacity, int * ip);
```

Arguments

sdev

scsi device

dev

block layer device

capacity

size in sectors

ip

geometry array

Description

This is anyone's guess quite frankly. We use the same rules everyone else appears to and hope. It seems to work.

i2o_scsi_init

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_scsi_init` — SCSI OSM initialization function

Synopsis

```
int i2o_scsi_init ( void);
```

Arguments

void

no arguments

Description

Register SCSI OSM into I2O core.

Returns 0 on success or negative error code on failure.

i2o_scsi_exit

LINUX

Name

`i2o_scsi_exit` — SCSI OSM exit function

Synopsis

```
void __exit i2o_scsi_exit ( void);
```

Arguments

void

no arguments

Description

Unregisters SCSI OSM from I2O core.

i2o_get_class_name

LINUX

Name

`i2o_get_class_name` — do i2o class name lookup

Synopsis

```
const char * i2o_get_class_name (int class);
```

Arguments

class

class number

Description

Return a descriptive string for an i2o class.

i2o_proc_create_entries

LINUX

Kernel Hackers ManualApril 2016

Name

i2o_proc_create_entries — Creates proc dir entries

Synopsis

```
int i2o_proc_create_entries (struct proc_dir_entry * dir,
i2o_proc_entry * i2o_pe, void * data);
```

Arguments

dir

proc dir entry under which the entries should be placed

i2o_pe

pointer to the entries which should be added

data

pointer to I2O controller or device

Description

Create proc dir entries for a I2O controller or I2O device.

Returns 0 on success or negative error code on failure.

i2o_proc_subdir_remove

LINUX

Kernel Hackers ManualApril 2016

Name

i2o_proc_subdir_remove — Remove child entries from a proc entry

Synopsis

```
void i2o_proc_subdir_remove (struct proc_dir_entry * dir);
```

Arguments

dir

proc dir entry from which the childs should be removed

Description

Iterate over each i2o proc entry under dir and remove it. If the child also has entries, remove them too.

i2o_proc_device_add

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_proc_device_add` — Add an I2O device to the proc dir

Synopsis

```
void i2o_proc_device_add (struct proc_dir_entry * dir, struct
i2o_device * dev);
```

Arguments

dir

proc dir entry to which the device should be added

dev

I2O device which should be added

Description

Add an I2O device to the proc dir entry dir and create the entries for the device depending on the class of the I2O device.

i2o_proc_iop_add

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_proc_iop_add` — Add an I2O controller to the i2o proc tree

Synopsis

```
int i2o_proc_iop_add (struct proc_dir_entry * dir, struct
i2o_controller * c);
```

Arguments

dir

parent proc dir entry

c

I2O controller which should be added

Description

Add the entries to the parent proc dir entry. Also each device is added to the controllers proc dir entry.

Returns 0 on success or negative error code on failure.

i2o_proc_iop_remove

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_proc_iop_remove` — Removes an I2O controller from the i2o proc tree

Synopsis

```
void i2o_proc_iop_remove (struct proc_dir_entry * dir, struct
i2o_controller * c);
```

Arguments

dir

parent proc dir entry

c

I2O controller which should be removed

Description

Iterate over each i2o proc entry and search controller c. If it is found remove it from the tree.

i2o_proc_fs_create

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_proc_fs_create` — Create the i2o proc fs.

Synopsis

```
int i2o_proc_fs_create ( void);
```

Arguments

void

no arguments

Description

Iterate over each I2O controller and create the entries for it.

Returns 0 on success or negative error code on failure.

i2o_proc_fs_destroy

LINUX

Name

`i2o_proc_fs_destroy` — Cleanup the all i2o proc entries

Synopsis

```
int __exit i2o_proc_fs_destroy ( void);
```

Arguments

`void`

no arguments

Description

Iterate over each I2O controller and remove the entries for it.

Returns 0 on success or negative error code on failure.

i2o_proc_init

LINUX

Name

`i2o_proc_init` — Init function for procfs

Synopsis

```
int i2o_proc_init ( void);
```

Arguments

void

no arguments

Description

Registers Proc OSM and creates procfs entries.

Returns 0 on success or negative error code on failure.

i2o_proc_exit

LINUX

Kernel Hackers ManualApril 2016

Name

`i2o_proc_exit` — Exit function for procfs

Synopsis

```
void __exit i2o_proc_exit ( void);
```

Arguments

void

no arguments

Description

Unregisters Proc OSM and removes procfs entries.

Chapter 5. Sound Devices

snd_register_device

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_register_device` — Register the ALSA device file for the card

Synopsis

```
int snd_register_device (int type, struct snd_card * card, int
dev, const struct file_operations * f_ops, void *
private_data, const char * name);
```

Arguments

`type`

the device type, SNDRV_DEVICE_TYPE_XXX

`card`

the card instance

`dev`

the device index

`f_ops`

the file operations

`private_data`

user pointer for f_ops->`open`

name

the device file name

Description

Registers an ALSA device file for the given card. The operators have to be set in reg parameter.

This function uses the card's device pointer to link to the correct struct device.

Returns zero if successful, or a negative error code on failure.

snd_printk

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_printk` — printk wrapper

Synopsis

```
snd_printk ( fmt,  args...);
```

Arguments

fmt

format string

args...

variable arguments

Description

Works like `printk` but prints the file and the line of the caller when configured with CONFIG_SND_VERBOSE_PRINTK.

snd_printd

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_printd` — debug printk

Synopsis

```
snd_printd ( fmt,  args...);
```

Arguments

`fmt`

format string

`args...`

variable arguments

Description

Works like `snd_printk` for debugging purposes. Ignored when CONFIG_SND_DEBUG is not set.

snd_BUG

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_BUG` — give a BUG warning message and stack trace

Synopsis

```
snd_BUG (void);
```

Arguments

None

Description

Calls `WARN` if CONFIG_SND_DEBUG is set. Ignored when CONFIG_SND_DEBUG is not set.

snd_BUG_ON

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_BUG_ON` — debugging check macro

Synopsis

```
snd_BUG_ON ( cond);
```

Arguments

cond

condition to evaluate

Description

When CONFIG_SND_DEBUG is set, this macro evaluates the given condition, and call `WARN` and returns the value if it's non-zero.

When CONFIG_SND_DEBUG is not set, this just returns zero, and the given condition is ignored.

NOTE

the argument won't be evaluated at all when CONFIG_SND_DEBUG=n. Thus, don't put any statement that influences on the code behavior, such as pre/post increment, to the argument of this macro. If you want to evaluate and give a warning, use standard `WARN_ON`.

snd_printdd

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_printdd` — debug printk

Synopsis

```
snd_printdd ( format,  args...);
```

Arguments

format

format string

args...

variable arguments

Description

Works like `snd_printk` for debugging purposes. Ignored when CONFIG_SND_DEBUG_VERBOSE is not set.

register_sound_special_device

LINUX

Kernel Hackers ManualApril 2016

Name

`register_sound_special_device` — register a special sound node

Synopsis

```
int register_sound_special_device (const struct
file_operations * fops, int unit, struct device * dev);
```

Arguments

fops

File operations for the driver

unit

Unit number to allocate

dev

device pointer

Description

Allocate a special sound device by minor number from the sound subsystem. The allocated number is returned on success. On failure a negative error code is returned.

register_sound_mixer

LINUX

Kernel Hackers ManualApril 2016

Name

`register_sound_mixer` — register a mixer device

Synopsis

```
int register_sound_mixer (const struct file_operations * fops,
int dev);
```

Arguments

fops

File operations for the driver

dev

Unit number to allocate

Description

Allocate a mixer device. Unit is the number of the mixer requested. Pass -1 to request the next free mixer unit. On success the allocated number is returned, on failure a negative error code is returned.

register_sound_midi

LINUX

Kernel Hackers ManualApril 2016

Name

`register_sound_midi` — register a midi device

Synopsis

```
int register_sound_midi (const struct file_operations * fops,
int dev);
```

Arguments

fops

File operations for the driver

dev

Unit number to allocate

Description

Allocate a midi device. Unit is the number of the midi device requested. Pass -1 to request the next free midi unit. On success the allocated number is returned, on failure a negative error code is returned.

register_sound_dsp

LINUX

Kernel Hackers ManualApril 2016

Name

`register_sound_dsp` — register a DSP device

Synopsis

```
int register_sound_dsp (const struct file_operations * fops,
int dev);
```

Arguments

fops

File operations for the driver

dev

Unit number to allocate

Description

Allocate a DSP device. Unit is the number of the DSP requested. Pass -1 to request the next free DSP unit. On success the allocated number is returned, on failure a negative error code is returned.

This function allocates both the audio and dsp device entries together and will always allocate them as a matching pair - eg dsp3/audio3

unregister_sound_special

LINUX

Kernel Hackers ManualApril 2016

Name

`unregister_sound_special` — unregister a special sound device

Synopsis

```
void unregister_sound_special (int unit);
```

Arguments

unit

unit number to allocate

Description

Release a sound device that was allocated with `register_sound_special`. The unit passed is the return value from the register function.

unregister_sound_mixer

LINUX

Kernel Hackers ManualApril 2016

Name

`unregister_sound_mixer` — unregister a mixer

Synopsis

```
void unregister_sound_mixer (int unit);
```

Arguments

unit

unit number to allocate

Description

Release a sound device that was allocated with `register_sound_mixer`. The unit passed is the return value from the register function.

unregister_sound_midi

LINUX

Kernel Hackers ManualApril 2016

Name

`unregister_sound_midi` — unregister a midi device

Synopsis

```
void unregister_sound_midi (int unit);
```

Arguments

unit

unit number to allocate

Description

Release a sound device that was allocated with `register_sound_midi`. The unit passed is the return value from the register function.

unregister_sound_dsp

LINUX

Kernel Hackers ManualApril 2016

Name

`unregister_sound_dsp` — unregister a DSP device

Synopsis

```
void unregister_sound_dsp (int unit);
```

Arguments

`unit`

unit number to allocate

Description

Release a sound device that was allocated with `register_sound_dsp`. The unit passed is the return value from the register function.

Both of the allocated units are released together automatically.

snd_pcm_playback_ready

LINUX

Name

`snd_pcm_playback_ready` — check whether the playback buffer is available

Synopsis

```
int snd_pcm_playback_ready (struct snd_pcm_substream *
substream);
```

Arguments

substream

the pcm substream instance

Description

Checks whether enough free space is available on the playback buffer.

Returns non-zero if available, or zero if not.

snd_pcm_capture_ready

LINUX

Name

`snd_pcm_capture_ready` — check whether the capture buffer is available

Synopsis

```
int snd_pcm_capture_ready (struct snd_pcm_substream *
substream);
```

Arguments

substream

the pcm substream instance

Description

Checks whether enough capture data is available on the capture buffer.

Returns non-zero if available, or zero if not.

snd_pcm_playback_data

LINUX

Kernel Hackers ManualApril 2016

Name

snd_pcm_playback_data — check whether any data exists on the playback buffer

Synopsis

```
int snd_pcm_playback_data (struct snd_pcm_substream *
substream);
```

Arguments

substream

the pcm substream instance

Description

Checks whether any data exists on the playback buffer. If stop_threshold is bigger or equal to boundary, then this function returns always non-zero.

Returns non-zero if exists, or zero if not.

snd_pcm_playback_empty

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_playback_empty` — check whether the playback buffer is empty

Synopsis

```
int snd_pcm_playback_empty (struct snd_pcm_substream *
substream);
```

Arguments

substream

the pcm substream instance

Description

Checks whether the playback buffer is empty.

Returns non-zero if empty, or zero if not.

snd_pcm_capture_empty

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_capture_empty` — check whether the capture buffer is empty

Synopsis

```
int snd_pcm_capture_empty (struct snd_pcm_substream *
substream);
```

Arguments

substream

the pcm substream instance

Description

Checks whether the capture buffer is empty.

Returns non-zero if empty, or zero if not.

snd_pcm_format_cpu_endian

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_format_cpu_endian` — Check the PCM format is CPU-endian

Synopsis

```
int snd_pcm_format_cpu_endian (snd_pcm_format_t format);
```

Arguments

`format`

the format to check

Description

Returns 1 if the given PCM format is CPU-endian, 0 if opposite, or a negative error code if endian not specified.

snd_pcm_lib_alloc_vmalloc_buffer

LINUX

Name

`snd_pcm_lib_alloc_vmalloc_buffer` — allocate virtual DMA buffer

Synopsis

```
int snd_pcm_lib_alloc_vmalloc_buffer (struct snd_pcm_substream
* substream, size_t size);
```

Arguments

substream

the substream to allocate the buffer to

size

the requested buffer size, in bytes

Description

Allocates the PCM substream buffer using `vmalloc`, i.e., the memory is contiguous in kernel virtual space, but not in physical memory. Use this if the buffer is accessed by kernel code but not by device DMA.

Returns 1 if the buffer was changed, 0 if not changed, or a negative error code.

snd_pcm_lib_alloc_vmalloc_32_buffer

LINUX

Name

`snd_pcm_lib_alloc_vmalloc_32_buffer` — allocate 32-bit-addressable buffer

Synopsis

```
int snd_pcm_lib_alloc_vmalloc_32_buffer (struct
snd_pcm_substream * substream, size_t size);
```

Arguments

`substream`

the substream to allocate the buffer to

`size`

the requested buffer size, in bytes

Description

This function works like `snd_pcm_lib_alloc_vmalloc_buffer`, but uses `vmalloc_32`, i.e., the pages are allocated from 32-bit-addressable memory.

snd_pcm_new_stream

LINUX

Name

`snd_pcm_new_stream` — create a new PCM stream

Synopsis

```
int snd_pcm_new_stream (struct snd_pcm * pcm, int stream, int
substream_count);
```

Arguments

pcm

the pcm instance

stream

the stream direction, SNDRV_PCM_STREAM_XXX

substream_count

the number of substreams

Description

Creates a new stream for the pcm. The corresponding stream on the pcm must have been empty before calling this, i.e. zero must be given to the argument of `snd_pcm_new`.

Returns zero if successful, or a negative error code on failure.

snd_pcm_new

LINUX

Name

`snd_pcm_new` — create a new PCM instance

Synopsis

```
int snd_pcm_new (struct snd_card * card, const char * id, int
device, int playback_count, int capture_count, struct snd_pcm
** rpcm);
```

Arguments

`card`

the card instance

`id`

the id string

`device`

the device index (zero based)

`playback_count`

the number of substreams for playback

`capture_count`

the number of substreams for capture

`rpcm`

the pointer to store the new pcm instance

Description

Creates a new PCM instance.

The pcm operators have to be set afterwards to the new instance via `snd_pcm_set_ops`.

Returns zero if successful, or a negative error code on failure.

snd_device_new

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_device_new` — create an ALSA device component

Synopsis

```
int snd_device_new (struct snd_card * card, snd_device_type_t
type, void * device_data, struct snd_device_ops * ops);
```

Arguments

`card`

the card instance

`type`

the device type, SNDRV_DEV_XXX

`device_data`

the data pointer of this device

`ops`

the operator table

Description

Creates a new device component for the given data pointer. The device will be assigned to the card and managed together by the card.

The data pointer plays a role as the identifier, too, so the pointer address must be unique and unchanged.

Returns zero if successful, or a negative error code on failure.

snd_device_free

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_device_free` — release the device from the card

Synopsis

```
int snd_device_free (struct snd_card * card, void *
device_data);
```

Arguments

`card`

the card instance

`device_data`

the data pointer to release

Description

Removes the device from the list on the card and invokes the callbacks, dev_disconnect and dev_free, corresponding to the state. Then release the device.

Returns zero if successful, or a negative error code on failure or if the device not found.

snd_device_register

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_device_register` — register the device

Synopsis

```
int snd_device_register (struct snd_card * card, void *
device_data);
```

Arguments

`card`

the card instance

`device_data`

the data pointer to register

Description

Registers the device which was already created via `snd_device_new`. Usually this is called from `snd_card_register`, but it can be called later if any new devices are created after invocation of `snd_card_register`.

Returns zero if successful, or a negative error code on failure or if the device not found.

snd_iprintf

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_iprintf` — printf on the procfs buffer

Synopsis

```
int snd_iprintf (struct snd_info_buffer * buffer, const char *
fmt,  ...);
```

Arguments

`buffer`

the procfs buffer

`fmt`

the printf format

`...`

variable arguments

Description

Outputs the string on the procfs buffer just like `printf`.

Returns the size of output string.

snd_info_get_line

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_info_get_line` — read one line from the procfs buffer

Synopsis

```
int snd_info_get_line (struct snd_info_buffer * buffer, char *
line, int len);
```

Arguments

`buffer`

the procfs buffer

`line`

the buffer to store

`len`

the max. buffer size - 1

Description

Reads one line from the buffer and stores the string.

Returns zero if successful, or 1 if error or EOF.

snd_info_get_str

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_info_get_str` — parse a string token

Synopsis

```
const char * snd_info_get_str (char * dest, const char * src,
int len);
```

Arguments

dest

the buffer to store the string token

src

the original string

len

the max. length of token - 1

Description

Parses the original string and copy a token to the given string buffer.

Returns the updated pointer of the original string so that it can be used for the next call.

snd_info_create_module_entry

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_info_create_module_entry` — create an info entry for the given module

Synopsis

```
struct snd_info_entry * snd_info_create_module_entry (struct
module * module, const char * name, struct snd_info_entry *
parent);
```

Arguments

module

the module pointer

name

the file name

parent

the parent directory

Description

Creates a new info entry and assigns it to the given module.

Returns the pointer of the new instance, or NULL on failure.

snd_info_create_card_entry

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_info_create_card_entry` — create an info entry for the given card

Synopsis

```
struct snd_info_entry * snd_info_create_card_entry (struct
snd_card * card, const char * name, struct snd_info_entry *
parent);
```

Arguments

card

the card instance

name

the file name

parent

the parent directory

Description

Creates a new info entry and assigns it to the given card.

Returns the pointer of the new instance, or NULL on failure.

snd_card_proc_new

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_card_proc_new` — create an info entry for the given card

Synopsis

```
int snd_card_proc_new (struct snd_card * card, const char *
name, struct snd_info_entry ** entryp);
```

Arguments

card

the card instance

name

the file name

entryp

the pointer to store the new info entry

Description

Creates a new info entry and assigns it to the given card. Unlike `snd_info_create_card_entry`, this function registers the info entry as an ALSA device component, so that it can be unregistered/released without explicit call. Also, you don't have to register this entry via `snd_info_register`, since this will be registered by `snd_card_register` automatically.

The parent is assumed as card->proc_root.

For releasing this entry, use `snd_device_free` instead of `snd_info_free_entry`.

Returns zero if successful, or a negative error code on failure.

snd_info_free_entry

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_info_free_entry` — release the info entry

Synopsis

```
void snd_info_free_entry (struct snd_info_entry * entry);
```

Arguments

`entry`

the info entry

Description

Releases the info entry. Don't call this after registered.

snd_info_register

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_info_register` — register the info entry

Synopsis

```
int snd_info_register (struct snd_info_entry * entry);
```

Arguments

entry

the info entry

Description

Registers the proc info entry.

Returns zero if successful, or a negative error code on failure.

snd_rawmidi_receive

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_rawmidi_receive` — receive the input data from the device

Synopsis

```
int snd_rawmidi_receive (struct snd_rawmidi_substream *
substream, const unsigned char * buffer, int count);
```

Arguments

substream

the rawmidi substream

buffer

the buffer pointer

count

the data size to read

Description

Reads the data from the internal buffer.

Returns the size of read data, or a negative error code on failure.

snd_rawmidi_transmit_empty

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_rawmidi_transmit_empty` — check whether the output buffer is empty

Synopsis

```
int snd_rawmidi_transmit_empty (struct snd_rawmidi_substream *
substream);
```

Arguments

substream

the rawmidi substream

Description

Returns 1 if the internal output buffer is empty, 0 if not.

snd_rawmidi_transmit_peek

LINUX

Name

`snd_rawmidi_transmit_peek` — copy data from the internal buffer

Synopsis

```
int snd_rawmidi_transmit_peek (struct snd_rawmidi_substream *
substream, unsigned char * buffer, int count);
```

Arguments

substream

the rawmidi substream

buffer

the buffer pointer

count

data size to transfer

Description

Copies data from the internal output buffer to the given buffer.

Call this in the interrupt handler when the midi output is ready, and call `snd_rawmidi_transmit_ack` after the transmission is finished.

Returns the size of copied data, or a negative error code on failure.

snd_rawmidi_transmit_ack

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_rawmidi_transmit_ack` — acknowledge the transmission

Synopsis

```
int snd_rawmidi_transmit_ack (struct snd_rawmidi_substream *
substream, int count);
```

Arguments

substream

the rawmidi substream

count

the tranferred count

Description

Advances the hardware pointer for the internal output buffer with the given size and updates the condition. Call after the transmission is finished.

Returns the advanced size if successful, or a negative error code on failure.

snd_rawmidi_transmit

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_rawmidi_transmit` — copy from the buffer to the device

Synopsis

```
int snd_rawmidi_transmit (struct snd_rawmidi_substream *
substream, unsigned char * buffer, int count);
```

Arguments

substream

the rawmidi substream

buffer

the buffer pointer

count

the data size to transfer

Description

Copies data from the buffer to the device and advances the pointer.

Returns the copied size if successful, or a negative error code on failure.

snd_rawmidi_new

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_rawmidi_new` — create a rawmidi instance

Synopsis

```
int snd_rawmidi_new (struct snd_card * card, char * id, int
device, int output_count, int input_count, struct snd_rawmidi
** rrawmidi);
```

Arguments

card

the card instance

id

the id string

device

the device index

output_count

the number of output streams

input_count

the number of input streams

rrawmidi

the pointer to store the new rawmidi instance

Description

Creates a new rawmidi instance. Use `snd_rawmidi_set_ops` to set the operators to the new instance.

Returns zero if successful, or a negative error code on failure.

snd_rawmidi_set_ops

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_rawmidi_set_ops` — set the rawmidi operators

Synopsis

```
void snd_rawmidi_set_ops (struct snd_rawmidi * rmidi, int
stream, struct snd_rawmidi_ops * ops);
```

Arguments

`rmidi`

the rawmidi instance

`stream`

the stream direction, SNDRV_RAWMIDI_STREAM_XXX

`ops`

the operator table

Description

Sets the rawmidi operators for the given stream direction.

snd_request_card

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_request_card` — try to load the card module

Synopsis

```
void snd_request_card (int card);
```

Arguments

`card`

the card number

Description

Tries to load the module “snd-card-X” for the given card number via request_module. Returns immediately if already loaded.

snd_lookup_minor_data

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_lookup_minor_data` — get user data of a registered device

Synopsis

```
void * snd_lookup_minor_data (unsigned int minor, int type);
```

Arguments

minor

the minor number

type

device type (SNDRV_DEVICE_TYPE_XXX)

Description

Checks that a minor device with the specified type is registered, and returns its user data pointer.

This function increments the reference counter of the card instance if an associated instance with the given minor number and type is found. The caller must call `snd_card_unref` appropriately later.

snd_register_device_for_dev

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_register_device_for_dev` — Register the ALSA device file for the card

Synopsis

```
int snd_register_device_for_dev (int type, struct snd_card * card, int dev, const struct file_operations * f_ops, void * private_data, const char * name, struct device * device);
```

Arguments

type

the device type, SNDRV_DEVICE_TYPE_XXX

card

the card instance

dev

the device index

f_ops

the file operations

private_data

user pointer for f_ops->`open`

name

the device file name

device

the struct device to link this new device to

Description

Registers an ALSA device file for the given card. The operators have to be set in reg parameter.

Returns zero if successful, or a negative error code on failure.

snd_unregister_device

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_unregister_device` — unregister the device on the given card

Synopsis

```
int snd_unregister_device (int type, struct snd_card * card,
int dev);
```

Arguments

type

the device type, SNDRV_DEVICE_TYPE_XXX

card

the card instance

dev

the device index

Description

Unregisters the device file already registered via `snd_register_device`.

Returns zero if sucecessful, or a negative error code on failure

copy_to_user_fromio

LINUX

Kernel Hackers ManualApril 2016

Name

`copy_to_user_fromio` — copy data from mmio-space to user-space

Synopsis

```
int copy_to_user_fromio (void __user * dst, const volatile
void __iomem * src, size_t count);
```

Arguments

dst

the destination pointer on user-space

src

the source pointer on mmio

count

the data size to copy in bytes

Description

Copies the data from mmio-space to user-space.

Returns zero if successful, or non-zero on failure.

copy_from_user_toio

LINUX

Kernel Hackers ManualApril 2016

Name

`copy_from_user_toio` — copy data from user-space to mmio-space

Synopsis

```
int copy_from_user_toio (volatile void __iomem * dst, const
void __user * src, size_t count);
```

Arguments

dst

the destination pointer on mmio-space

src

the source pointer on user-space

count

the data size to copy in bytes

Description

Copies the data from user-space to mmio-space.

Returns zero if successful, or non-zero on failure.

snd_pcm_lib_preallocate_free_for_all

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_lib_preallocate_free_for_all` — release all pre-allocated buffers on the pcm

Synopsis

```
int snd_pcm_lib_preallocate_free_for_all (struct snd_pcm * pcm);
```

Arguments

pcm

the pcm instance

Description

Releases all the pre-allocated buffers on the given pcm.

Returns zero if successful, or a negative error code on failure.

snd_pcm_lib_preallocate_pages

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_lib_preallocate_pages` — pre-allocation for the given DMA type

Synopsis

```
int snd_pcm_lib_preallocate_pages (struct snd_pcm_substream * substream, int type, struct device * data, size_t size, size_t max);
```

Arguments

substream

the pcm substream instance

type

DMA type (SNDRV_DMA_TYPE_*)

data

DMA type dependent data

size

the requested pre-allocation size in bytes

max

the max. allowed pre-allocation size

Description

Do pre-allocation for the given DMA buffer type.

When substream->dma_buf_id is set, the function tries to look for the reserved buffer, and the buffer is not freed but reserved at destruction time. The dma_buf_id must be unique for all systems (in the same DMA buffer type) e.g. using `snd_dma_pci_buf_id`.

Returns zero if successful, or a negative error code on failure.

snd_pcm_lib_preallocate_pages_for_all

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_lib_preallocate_pages_for_all` — pre-allocation for continuous memory type (all substreams)

Synopsis

```
int snd_pcm_lib_preallocate_pages_for_all (struct snd_pcm *
pcm, int type, void * data, size_t size, size_t max);
```

Arguments

pcm

the pcm instance

type

DMA type (SNDRV_DMA_TYPE_*)

data

DMA type dependent data

size

the requested pre-allocation size in bytes

max

the max. allowed pre-allocation size

Description

Do pre-allocation to all substreams of the given pcm for the specified DMA type.

Returns zero if successful, or a negative error code on failure.

snd_pcm_sgbuf_ops_page

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_sgbuf_ops_page` — get the page struct at the given offset

Synopsis

```
struct page * snd_pcm_sgbuf_ops_page (struct snd_pcm_substream
* substream, unsigned long offset);
```

Arguments

substream

the pcm substream instance

offset

the buffer offset

Description

Returns the page struct at the given buffer offset. Used as the page callback of PCM ops.

snd_pcm_lib_malloc_pages

LINUX

Kernel Hackers ManualApril 2016

Name

snd_pcm_lib_malloc_pages — allocate the DMA buffer

Synopsis

```
int snd_pcm_lib_malloc_pages (struct snd_pcm_substream *
substream, size_t size);
```

Arguments

substream

the substream to allocate the DMA buffer to

size

the requested buffer size in bytes

Description

Allocates the DMA buffer on the BUS type given earlier to `snd_pcm_lib_preallocate_xxx_pages`.

Returns 1 if the buffer is changed, 0 if not changed, or a negative code on failure.

snd_pcm_lib_free_pages

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_lib_free_pages` — release the allocated DMA buffer.

Synopsis

```
int snd_pcm_lib_free_pages (struct snd_pcm_substream *
substream);
```

Arguments

substream

the substream to release the DMA buffer

Description

Releases the DMA buffer allocated via `snd_pcm_lib_malloc_pages`.

Returns zero if successful, or a negative error code on failure.

snd_pcm_lib_free_vmalloc_buffer

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_lib_free_vmalloc_buffer` — free vmalloc buffer

Synopsis

```
int snd_pcm_lib_free_vmalloc_buffer (struct snd_pcm_substream
* substream);
```

Arguments

substream

the substream with a buffer allocated by `snd_pcm_lib_alloc_vmalloc_buffer`

snd_pcm_lib_get_vmalloc_page

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_lib_get_vmalloc_page` — map vmalloc buffer offset to page struct

Synopsis

```
struct page * snd_pcm_lib_get_vmalloc_page (struct
snd_pcm_substream * substream, unsigned long offset);
```

Arguments

substream

the substream with a buffer allocated by `snd_pcm_lib_alloc_vmalloc_buffer`

offset

offset in the buffer

Description

This function is to be used as the page callback in the PCM ops.

snd_card_create

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_card_create` — create and initialize a soundcard structure

Synopsis

```
int snd_card_create (int idx, const char * xid, struct module
* module, int extra_size, struct snd_card ** card_ret);
```

Arguments

idx

card index (address) [0 ... (SNDRV_CARDS-1)]

xid

card identification (ASCII string)

module

top level module for locking

extra_size

allocate this extra size after the main soundcard structure

card_ret

the pointer to store the created card instance

Description

Creates and initializes a soundcard structure.

The function allocates snd_card instance via kzalloc with the given space for the driver to use freely. The allocated struct is stored in the given card_ret pointer.

Returns zero if successful or a negative error code.

snd_card_disconnect

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_card_disconnect` — disconnect all APIs from the file-operations (user space)

Synopsis

```
int snd_card_disconnect (struct snd_card * card);
```

Arguments

`card`

soundcard structure

Description

Disconnects all APIs from the file-operations (user space).

Returns zero, otherwise a negative error code.

Note

The current implementation replaces all active file->f_op with special dummy file operations (they do nothing except release).

snd_card_unref

Name

`snd_card_unref` — release the reference counter

Synopsis

```
void snd_card_unref (struct snd_card * card);
```

Arguments

`card`

the card instance

Description

Decrements the reference counter. When it reaches to zero, wake up the sleeper and call the destructor if needed.

snd_card_set_id

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_card_set_id` — set card identification name

Synopsis

```
void snd_card_set_id (struct snd_card * card, const char *
nid);
```

Arguments

card

soundcard structure

nid

new identification string

Description

This function sets the card identification and checks for name collisions.

snd_card_register

LINUX

Name

`snd_card_register` — register the soundcard

Synopsis

```
int snd_card_register (struct snd_card * card);
```

Arguments

`card`

soundcard structure

Description

This function registers all the devices assigned to the soundcard. Until calling this, the ALSA control interface is blocked from the external accesses. Thus, you should call this function at the end of the initialization of the card.

Returns zero otherwise a negative error code if the registration failed.

snd_component_add

LINUX

Name

`snd_component_add` — add a component string

Synopsis

```
int snd_component_add (struct snd_card * card, const char *
component);
```

Arguments

card

soundcard structure

component

the component id string

Description

This function adds the component id string to the supported list. The component can be referred from the alsa-lib.

Returns zero otherwise a negative error code.

snd_card_file_add

LINUX

Kernel Hackers ManualApril 2016

Name

snd_card_file_add — add the file to the file list of the card

Synopsis

```
int snd_card_file_add (struct snd_card * card, struct file *
file);
```

Arguments

card

soundcard structure

file

file pointer

Description

This function adds the file to the file linked-list of the card. This linked-list is used to keep tracking the connection state, and to avoid the release of busy resources by hotplug.

Returns zero or a negative error code.

snd_card_file_remove

LINUX

Kernel Hackers ManualApril 2016

Name

snd_card_file_remove — remove the file from the file list

Synopsis

```
int snd_card_file_remove (struct snd_card * card, struct file
* file);
```

Arguments

card

soundcard structure

file

file pointer

Description

This function removes the file formerly added to the card via `snd_card_file_add` function. If all files are removed and `snd_card_free_when_closed` was called beforehand, it processes the pending release of resources.

Returns zero or a negative error code.

snd_power_wait

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_power_wait` — wait until the power-state is changed.

Synopsis

```
int snd_power_wait (struct snd_card * card, unsigned int
power_state);
```

Arguments

card

soundcard structure

power_state

expected power state

Description

Waits until the power-state is changed.

Note

the power lock must be active before call.

snd_dma_program

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_dma_program` — program an ISA DMA transfer

Synopsis

```
void snd_dma_program (unsigned long dma, unsigned long addr,
unsigned int size, unsigned short mode);
```

Arguments

dma

the dma number

addr

the physical address of the buffer

size

the DMA transfer size

mode

the DMA transfer mode, DMA_MODE_XXX

Description

Programs an ISA DMA transfer for the given buffer.

snd_dma_disable

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_dma_disable` — stop the ISA DMA transfer

Synopsis

```
void snd_dma_disable (unsigned long dma);
```

Arguments

dma

the dma number

Description

Stops the ISA DMA transfer.

snd_dma_pointer

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_dma_pointer` — return the current pointer to DMA transfer buffer in bytes

Synopsis

```
unsigned int snd_dma_pointer (unsigned long dma, unsigned int
size);
```

Arguments

dma

the dma number

size

the dma transfer size

Description

Returns the current pointer in DMA tranfer buffer in bytes

snd_ctl_new1

LINUX

Kernel Hackers ManualApril 2016

Name

snd_ctl_new1 — create a control instance from the template

Synopsis

```
struct snd_kcontrol * snd_ctl_new1 (const struct
snd_kcontrol_new * ncontrol, void * private_data);
```

Arguments

ncontrol

the initialization record

private_data

the private data to set

Description

Allocates a new struct snd_kcontrol instance and initialize from the given template. When the access field of ncontrol is 0, it's assumed as READWRITE access. When the count field is 0, it's assumes as one.

Returns the pointer of the newly generated instance, or NULL on failure.

snd_ctl_free_one

LINUX

Kernel Hackers ManualApril 2016

Name

snd_ctl_free_one — release the control instance

Synopsis

```
void snd_ctl_free_one (struct snd_kcontrol * kcontrol);
```

Arguments

kcontrol

the control instance

Description

Releases the control instance created via `snd_ctl_new` or `snd_ctl_new1`. Don't call this after the control was added to the card.

snd_ctl_add

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_ctl_add` — add the control instance to the card

Synopsis

```
int snd_ctl_add (struct snd_card * card, struct snd_kcontrol *
kcontrol);
```

Arguments

`card`

the card instance

`kcontrol`

the control instance to add

Description

Adds the control instance created via `snd_ctl_new` or `snd_ctl_new1` to the given card. Assigns also an unique numid used for fast search.

Returns zero if successful, or a negative error code on failure.

It frees automatically the control which cannot be added.

snd_ctl_replace

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_ctl_replace` — replace the control instance of the card

Synopsis

```
int snd_ctl_replace (struct snd_card * card, struct
snd_kcontrol * kcontrol, bool add_on_replace);
```

Arguments

card

the card instance

kcontrol

the control instance to replace

add_on_replace

add the control if not already added

Description

Replaces the given control. If the given control does not exist and the add_on_replace flag is set, the control is added. If the control exists, it is destroyed

first.

Returns zero if successful, or a negative error code on failure.

It frees automatically the control which cannot be added or replaced.

snd_ctl_remove

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_ctl_remove` — remove the control from the card and release it

Synopsis

```
int snd_ctl_remove (struct snd_card * card, struct
snd_kcontrol * kcontrol);
```

Arguments

card

the card instance

kcontrol

the control instance to remove

Description

Removes the control from the card and then releases the instance. You don't need to call `snd_ctl_free_one`. You must be in the write lock - down_write(card->controls_rwsem).

Returns 0 if successful, or a negative error code on failure.

snd_ctl_remove_id

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_ctl_remove_id` — remove the control of the given id and release it

Synopsis

```
int snd_ctl_remove_id (struct snd_card * card, struct
snd_ctl_elem_id * id);
```

Arguments

card

the card instance

id

the control id to remove

Description

Finds the control instance with the given id, removes it from the card list and releases it.

Returns 0 if successful, or a negative error code on failure.

snd_ctl_activate_id

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_ctl_activate_id` — activate/inactivate the control of the given id

Synopsis

```
int snd_ctl_activate_id (struct snd_card * card, struct
snd_ctl_elem_id * id, int active);
```

Arguments

card

the card instance

id

the control id to activate/inactivate

active

non-zero to activate

Description

Finds the control instance with the given id, and activate or inactivate the control together with notification, if changed.

Returns 0 if unchanged, 1 if changed, or a negative error code on failure.

snd_ctl_rename_id

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_ctl_rename_id` — replace the id of a control on the card

Synopsis

```
int snd_ctl_rename_id (struct snd_card * card, struct
snd_ctl_elem_id * src_id, struct snd_ctl_elem_id * dst_id);
```

Arguments

card

the card instance

src_id

the old id

dst_id

the new id

Description

Finds the control with the old id from the card, and replaces the id with the new one.

Returns zero if successful, or a negative error code on failure.

snd_ctl_find_numid

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_ctl_find_numid` — find the control instance with the given number-id

Synopsis

```
struct snd_kcontrol * snd_ctl_find_numid (struct snd_card *
card, unsigned int numid);
```

Arguments

card

the card instance

numid

the number-id to search

Description

Finds the control instance with the given number-id from the card.

Returns the pointer of the instance if found, or NULL if not.

The caller must down card->controls_rwsem before calling this function (if the race condition can happen).

snd_ctl_find_id

LINUX

Kernel Hackers ManualApril 2016

Name

snd_ctl_find_id — find the control instance with the given id

Synopsis

```
struct snd_kcontrol * snd_ctl_find_id (struct snd_card * card,
struct snd_ctl_elem_id * id);
```

Arguments

card

the card instance

id

the id to search

Description

Finds the control instance with the given id from the card.

Returns the pointer of the instance if found, or NULL if not.

The caller must down card->controls_rwsem before calling this function (if the race condition can happen).

snd_ctl_enum_info

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_ctl_enum_info` — fills the info structure for an enumerated control

Synopsis

```
int snd_ctl_enum_info (struct snd_ctl_elem_info * info,
unsigned int channels, unsigned int items, const char *const
names[]);
```

Arguments

`info`

the structure to be filled

`channels`

the number of the control's channels; often one

`items`

the number of control values; also the size of *`names`*

`names[]`

an array containing the names of all control values

Description

Sets all required fields in *`info`* to their appropriate values. If the control's accessibility is not the default (readable and writable), the caller has to fill *`info`*->access.

snd_pcm_set_ops

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_set_ops` — set the PCM operators

Synopsis

```
void snd_pcm_set_ops (struct snd_pcm * pcm, int direction,
struct snd_pcm_ops * ops);
```

Arguments

pcm

the pcm instance

direction

stream direction, SNDRV_PCM_STREAM_XXX

ops

the operator table

Description

Sets the given PCM operators to the pcm instance.

snd_pcm_set_sync

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_set_sync` — set the PCM sync id

Synopsis

```
void snd_pcm_set_sync (struct snd_pcm_substream * substream);
```

Arguments

`substream`

the pcm substream

Description

Sets the PCM sync identifier for the card.

snd_interval_refine

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_interval_refine` — refine the interval value of configurator

Synopsis

```
int snd_interval_refine (struct snd_interval * i, const struct
snd_interval * v);
```

Arguments

i

the interval value to refine

v

the interval value to refer to

Description

Refines the interval value with the reference value. The interval is changed to the range satisfying both intervals. The interval status (min, max, integer, etc.) are evaluated.

Returns non-zero if the value is changed, zero if not changed.

snd_interval_ratnum

LINUX

Kernel Hackers ManualApril 2016

Name

snd_interval_ratnum — refine the interval value

Synopsis

```
int snd_interval_ratnum (struct snd_interval * i, unsigned int
rats_count, struct snd_ratnum * rats, unsigned int * nump,
unsigned int * denp);
```

Arguments

i

interval to refine

rats_count

number of ratnum_t

rats

ratnum_t array

nump

pointer to store the resultant numerator

denp

pointer to store the resultant denominator

Description

Returns non-zero if the value is changed, zero if not changed.

snd_interval_list

LINUX

Name

`snd_interval_list` — refine the interval value from the list

Synopsis

```
int snd_interval_list (struct snd_interval * i, unsigned int
count, unsigned int * list, unsigned int mask);
```

Arguments

i

the interval value to refine

count

the number of elements in the list

list

the value list

mask

the bit-mask to evaluate

Description

Refines the interval value from the list. When mask is non-zero, only the elements corresponding to bit 1 are evaluated.

Returns non-zero if the value is changed, zero if not changed.

snd_pcm_hw_rule_add

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_hw_rule_add` — add the hw-constraint rule

Synopsis

```
int snd_pcm_hw_rule_add (struct snd_pcm_runtime * runtime,
unsigned int cond, int var, snd_pcm_hw_rule_func_t func, void
* private, int dep,  ...);
```

Arguments

runtime

the pcm runtime instance

cond

condition bits

var

the variable to evaluate

func

the evaluation function

private

the private data pointer passed to function

dep

the dependent variables

...

variable arguments

Description

Returns zero if successful, or a negative error code on failure.

snd_pcm_hw_constraint_integer

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_hw_constraint_integer` — apply an integer constraint to an interval

Synopsis

```
int snd_pcm_hw_constraint_integer (struct snd_pcm_runtime *
runtime, snd_pcm_hw_param_t var);
```

Arguments

runtime

PCM runtime instance

var

hw_params variable to apply the integer constraint

Description

Apply the constraint of integer to an interval parameter.

snd_pcm_hw_constraint_minmax

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_hw_constraint_minmax` — apply a min/max range constraint to an interval

Synopsis

```
int snd_pcm_hw_constraint_minmax (struct snd_pcm_runtime *
runtime, snd_pcm_hw_param_t var, unsigned int min, unsigned
int max);
```

Arguments

runtime

PCM runtime instance

var

hw_params variable to apply the range

min

the minimal value

max

the maximal value

Description

Apply the min/max range constraint to an interval parameter.

snd_pcm_hw_constraint_list

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_hw_constraint_list` — apply a list of constraints to a parameter

Synopsis

```
int snd_pcm_hw_constraint_list (struct snd_pcm_runtime *
runtime, unsigned int cond, snd_pcm_hw_param_t var, struct
snd_pcm_hw_constraint_list * l);
```

Arguments

runtime

PCM runtime instance

cond

condition bits

var

hw_params variable to apply the list constraint

l

list

Description

Apply the list of constraints to an interval parameter.

snd_pcm_hw_constraint_ratnums

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_hw_constraint_ratnums` — apply ratnums constraint to a parameter

Synopsis

```
int snd_pcm_hw_constraint_ratnums (struct snd_pcm_runtime * runtime, unsigned int cond, snd_pcm_hw_param_t var, struct snd_pcm_hw_constraint_ratnums * r);
```

Arguments

runtime

PCM runtime instance

cond

condition bits

var

hw_params variable to apply the ratnums constraint

r

struct snd_ratnums constriants

snd_pcm_hw_constraint_ratdens

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_hw_constraint_ratdens` — apply ratdens constraint to a parameter

Synopsis

```
int snd_pcm_hw_constraint_ratdens (struct snd_pcm_runtime * runtime, unsigned int cond, snd_pcm_hw_param_t var, struct snd_pcm_hw_constraint_ratdens * r);
```

Arguments

runtime

PCM runtime instance

cond

condition bits

var

hw_params variable to apply the ratdens constraint

r

struct snd_ratdens constriants

snd_pcm_hw_constraint_msbits

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_hw_constraint_msbits` — add a hw constraint msbits rule

Synopsis

```
int snd_pcm_hw_constraint_msbits (struct snd_pcm_runtime * runtime, unsigned int cond, unsigned int width, unsigned int msbits);
```

Arguments

runtime

PCM runtime instance

cond

condition bits

width

sample bits width

msbits

msbits width

snd_pcm_hw_constraint_step

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_hw_constraint_step` — add a hw constraint step rule

Synopsis

```
int snd_pcm_hw_constraint_step (struct snd_pcm_runtime *
runtime, unsigned int cond, snd_pcm_hw_param_t var, unsigned
long step);
```

Arguments

runtime

PCM runtime instance

cond

condition bits

var

hw_params variable to apply the step constraint

step

step size

snd_pcm_hw_constraint_pow2

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_hw_constraint_pow2` — add a hw constraint power-of-2 rule

Synopsis

```
int snd_pcm_hw_constraint_pow2 (struct snd_pcm_runtime *
runtime, unsigned int cond, snd_pcm_hw_param_t var);
```

Arguments

runtime

PCM runtime instance

cond

condition bits

var

hw_params variable to apply the power-of-2 constraint

snd_pcm_hw_param_value

LINUX

Name

`snd_pcm_hw_param_value` — return *`params`* field *`var`* value

Synopsis

```
int snd_pcm_hw_param_value (const struct snd_pcm_hw_params *
params, snd_pcm_hw_param_t var, int * dir);
```

Arguments

`params`

the hw_params instance

`var`

parameter to retrieve

`dir`

pointer to the direction (-1,0,1) or `NULL`

Description

Return the value for field *`var`* if it's fixed in configuration space defined by *`params`*. Return -`EINVAL` otherwise.

snd_pcm_hw_param_first

LINUX

Name

snd_pcm_hw_param_first — refine config space and return minimum value

Synopsis

```
int snd_pcm_hw_param_first (struct snd_pcm_substream * pcm,
struct snd_pcm_hw_params * params, snd_pcm_hw_param_t var, int
* dir);
```

Arguments

pcm

PCM instance

params

the hw_params instance

var

parameter to retrieve

dir

pointer to the direction (-1,0,1) or NULL

Description

Inside configuration space defined by *params* remove from *var* all values > minimum. Reduce configuration space accordingly. Return the minimum.

snd_pcm_hw_param_last

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_hw_param_last` — refine config space and return maximum value

Synopsis

```
int snd_pcm_hw_param_last (struct snd_pcm_substream * pcm,
struct snd_pcm_hw_params * params, snd_pcm_hw_param_t var, int
* dir);
```

Arguments

pcm

PCM instance

params

the hw_params instance

var

parameter to retrieve

dir

pointer to the direction (-1,0,1) or `NULL`

Description

Inside configuration space defined by *params* remove from *var* all values < maximum. Reduce configuration space accordingly. Return the maximum.

snd_pcm_lib_ioctl

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_lib_ioctl` — a generic PCM ioctl callback

Synopsis

```
int snd_pcm_lib_ioctl (struct snd_pcm_substream * substream,
unsigned int cmd, void * arg);
```

Arguments

substream

the pcm substream instance

cmd

ioctl command

arg

ioctl argument

Description

Processes the generic ioctl commands for PCM. Can be passed as the ioctl callback for PCM ops.

Returns zero if successful, or a negative error code on failure.

snd_pcm_period_elapsed

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_pcm_period_elapsed` — update the pcm status for the next period

Synopsis

```
void snd_pcm_period_elapsed (struct snd_pcm_substream *
substream);
```

Arguments

substream

the pcm substream instance

Description

This function is called from the interrupt handler when the PCM has processed the period size. It will update the current pointer, wake up sleepers, etc.

Even if more than one periods have elapsed since the last call, you have to call this only once.

snd_hwdep_new

LINUX

Name

snd_hwdep_new — create a new hwdep instance

Synopsis

```
int snd_hwdep_new (struct snd_card * card, char * id, int
device, struct snd_hwdep ** rhwdep);
```

Arguments

card

the card instance

id

the id string

device

the device index (zero-based)

rhwdep

the pointer to store the new hwdep instance

Description

Creates a new hwdep instance with the given index on the card. The callbacks (hwdep->ops) must be set on the returned instance after this call manually by the caller.

Returns zero if successful, or a negative error code on failure.

snd_pcm_stop

LINUX

Kernel Hackers ManualApril 2016

Name

snd_pcm_stop — try to stop all running streams in the substream group

Synopsis

```
int snd_pcm_stop (struct snd_pcm_substream * substream,
snd_pcm_state_t state);
```

Arguments

substream

the PCM substream instance

state

PCM state after stopping the stream

Description

The state of each stream is then changed to the given state unconditionally.

snd_pcm_suspend

LINUX

Name

`snd_pcm_suspend` — trigger SUSPEND to all linked streams

Synopsis

```
int snd_pcm_suspend (struct snd_pcm_substream * substream);
```

Arguments

`substream`

the PCM substream

Description

After this call, all streams are changed to SUSPENDED state.

snd_pcm_suspend_all

LINUX

Name

`snd_pcm_suspend_all` — trigger SUSPEND to all substreams in the given pcm

Synopsis

```
int snd_pcm_suspend_all (struct snd_pcm * pcm);
```

Arguments

pcm

the PCM instance

Description

After this call, all streams are changed to SUSPENDED state.

snd_malloc_pages

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_malloc_pages` — allocate pages with the given size

Synopsis

```
void * snd_malloc_pages (size_t size, gfp_t gfp_flags);
```

Arguments

size

the size to allocate in bytes

gfp_flags

the allocation conditions, GFP_XXX

Description

Allocates the physically contiguous pages with the given size.

Returns the pointer of the buffer, or NULL if no enoguh memory.

snd_free_pages

LINUX

Kernel Hackers ManualApril 2016

Name

snd_free_pages — release the pages

Synopsis

```
void snd_free_pages (void * ptr, size_t size);
```

Arguments

ptr

the buffer pointer to release

size

the allocated buffer size

Description

Releases the buffer allocated via `snd_malloc_pages`.

snd_dma_alloc_pages

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_dma_alloc_pages` — allocate the buffer area according to the given type

Synopsis

```
int snd_dma_alloc_pages (int type, struct device * device,
size_t size, struct snd_dma_buffer * dmab);
```

Arguments

type

the DMA buffer type

device

the device pointer

size

the buffer size to allocate

dmab

buffer allocation record to store the allocated data

Description

Calls the memory-allocator function for the corresponding buffer type.

Returns zero if the buffer with the given size is allocated successfuly, other a negative value at error.

snd_dma_alloc_pages_fallback

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_dma_alloc_pages_fallback` — allocate the buffer area according to the given type with fallback

Synopsis

```
int snd_dma_alloc_pages_fallback (int type, struct device * device, size_t size, struct snd_dma_buffer * dmab);
```

Arguments

type

the DMA buffer type

device

the device pointer

size

the buffer size to allocate

dmab

buffer allocation record to store the allocated data

Description

Calls the memory-allocator function for the corresponding buffer type. When no space is left, this function reduces the size and tries to allocate again. The size actually allocated is stored in res_size argument.

Returns zero if the buffer with the given size is allocated successfuly, other a negative value at error.

snd_dma_free_pages

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_dma_free_pages` — release the allocated buffer

Synopsis

```
void snd_dma_free_pages (struct snd_dma_buffer * dmab);
```

Arguments

dmab

the buffer allocation record to release

Description

Releases the allocated buffer via `snd_dma_alloc_pages`.

snd_dma_get_reserved_buf

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_dma_get_reserved_buf` — get the reserved buffer for the given device

Synopsis

```
size_t snd_dma_get_reserved_buf (struct snd_dma_buffer * dmab,
unsigned int id);
```

Arguments

dmab

the buffer allocation record to store

id

the buffer id

Description

Looks for the reserved-buffer list and re-uses if the same buffer is found in the list. When the buffer is found, it's removed from the free list.

Returns the size of buffer if the buffer is found, or zero if not found.

snd_dma_reserve_buf

LINUX

Kernel Hackers ManualApril 2016

Name

`snd_dma_reserve_buf` — reserve the buffer

Synopsis

```
int snd_dma_reserve_buf (struct snd_dma_buffer * dmab,
unsigned int id);
```

Arguments

dmab

the buffer to reserve

id

the buffer id

Description

Reserves the given buffer as a reserved buffer.

Returns zero if successful, or a negative code at error.

Chapter 6. 16x50 UART Driver

/usr/src/linux-3.0.101-1//include/linux/serial_core.h

Name

`/usr/src/linux-3.0.101-1//include/linux/serial_core.h` — Document generation inconsistency

Oops

Warning

The template for this document tried to insert the structured comment from the file `/usr/src/linux-3.0.101-1//include/linux/serial_core.h` at this point, but none was found. This dummy section is inserted to allow generation to continue.

uart_update_timeout

LINUX

Kernel Hackers ManualApril 2016

Name

`uart_update_timeout` — update per-port FIFO timeout.

Synopsis

```
void uart_update_timeout (struct uart_port * port, unsigned
int cflag, unsigned int baud);
```

Arguments

port

uart_port structure describing the port

cflag

termios cflag value

baud

speed of the port

Description

Set the port FIFO timeout value. The *cflag* value should reflect the actual hardware settings.

uart_get_baud_rate

LINUX

Kernel Hackers ManualApril 2016

Name

uart_get_baud_rate — return baud rate for a particular port

Synopsis

```
unsigned int uart_get_baud_rate (struct uart_port * port,
struct ktermios * termios, struct ktermios * old, unsigned int
min, unsigned int max);
```

Arguments

port

uart_port structure describing the port in question.

termios

desired termios settings.

old

old termios (or NULL)

min

minimum acceptable baud rate

max

maximum acceptable baud rate

Description

Decode the termios structure into a numeric baud rate, taking account of the magic 38400 baud rate (with spd_* flags), and mapping the `B0` rate to 9600 baud.

If the new baud rate is invalid, try the old termios setting. If it's still invalid, we try 9600 baud.

Update the *termios* structure to reflect the baud rate we're actually going to be using. Don't do this for the case where B0 is requested ("hang up").

uart_get_divisor

LINUX

Kernel Hackers ManualApril 2016

Name

`uart_get_divisor` — return uart clock divisor

Synopsis

```
unsigned int uart_get_divisor (struct uart_port * port,
unsigned int baud);
```

Arguments

port

uart_port structure describing the port.

baud

desired baud rate

Description

Calculate the uart clock divisor for the port.

uart_parse_options

LINUX

Name

`uart_parse_options` — Parse serial port baud/parity/bits/flow contro.

Synopsis

```
void uart_parse_options (char * options, int * baud, int *
parity, int * bits, int * flow);
```

Arguments

options

pointer to option string

baud

pointer to an 'int' variable for the baud rate.

parity

pointer to an 'int' variable for the parity.

bits

pointer to an 'int' variable for the number of data bits.

flow

pointer to an 'int' variable for the flow control character.

Description

uart_parse_options decodes a string containing the serial console options. The format of the string is <baud><parity><bits><flow>,

eg

115200n8r

uart_set_options

LINUX

Kernel Hackers ManualApril 2016

Name

`uart_set_options` — setup the serial console parameters

Synopsis

```
int uart_set_options (struct uart_port * port, struct console
* co, int baud, int parity, int bits, int flow);
```

Arguments

port

pointer to the serial ports uart_port structure

co

console pointer

baud

baud rate

parity

parity character - 'n' (none), 'o' (odd), 'e' (even)

bits

number of data bits

flow

flow control character - 'r' (rts)

uart_register_driver

LINUX

Kernel Hackers ManualApril 2016

Name

`uart_register_driver` — register a driver with the uart core layer

Synopsis

```
int uart_register_driver (struct uart_driver * drv);
```

Arguments

drv

low level driver structure

Description

Register a uart driver with the core driver. We in turn register with the tty layer, and initialise the core driver per-port state.

We have a proc file in /proc/tty/driver which is named after the normal driver.

drv->port should be NULL, and the per-port structures should be registered using uart_add_one_port after this call has succeeded.

uart_unregister_driver

LINUX

Kernel Hackers ManualApril 2016

Name

`uart_unregister_driver` — remove a driver from the uart core layer

Synopsis

```
void uart_unregister_driver (struct uart_driver * drv);
```

Arguments

drv

low level driver structure

Description

Remove all references to a driver from the core driver. The low level driver must have removed all its ports via the `uart_remove_one_port` if it registered them with `uart_add_one_port`. (ie, drv->port == NULL)

uart_add_one_port

LINUX

Kernel Hackers ManualApril 2016

Name

uart_add_one_port — attach a driver-defined port structure

Synopsis

```
int uart_add_one_port (struct uart_driver * drv, struct
uart_port * uport);
```

Arguments

drv

pointer to the uart low level driver structure for this port

uport

uart port structure to use for this port.

Description

This allows the driver to register its own uart_port structure with the core driver. The main purpose is to allow the low level uart drivers to expand uart_port, rather than having yet more levels of structures.

uart_remove_one_port

LINUX

Name

`uart_remove_one_port` — detach a driver defined port structure

Synopsis

```
int uart_remove_one_port (struct uart_driver * drv, struct
uart_port * uport);
```

Arguments

drv

pointer to the uart low level driver structure for this port

uport

uart port structure for this port

Description

This unhooks (and hangs up) the specified port structure from the core driver. No further calls will be made to the low-level code for this port.

uart_handle_dcd_change

LINUX

Name

uart_handle_dcd_change — handle a change of carrier detect state

Synopsis

```
void uart_handle_dcd_change (struct uart_port * uport,
unsigned int status);
```

Arguments

uport

uart_port structure for the open port

status

new carrier detect status, nonzero if active

uart_handle_cts_change

LINUX

Name

uart_handle_cts_change — handle a change of clear-to-send state

Synopsis

```
void uart_handle_cts_change (struct uart_port * uport,
unsigned int status);
```

Arguments

uport

uart_port structure for the open port

status

new clear to send status, nonzero if active

serial8250_suspend_port

LINUX

Kernel Hackers ManualApril 2016

Name

serial8250_suspend_port — suspend one serial port

Synopsis

```
void serial8250_suspend_port (int line);
```

Arguments

line

serial line number

Description

Suspend one serial port.

serial8250_resume_port

LINUX

Kernel Hackers ManualApril 2016

Name

`serial8250_resume_port` — resume one serial port

Synopsis

```
void serial8250_resume_port (int line);
```

Arguments

line

serial line number

Description

Resume one serial port.

serial8250_register_port

LINUX

Kernel Hackers ManualApril 2016

Name

`serial8250_register_port` — register a serial port

Synopsis

```
int serial8250_register_port (struct uart_port * port);
```

Arguments

port

serial port template

Description

Configure the serial port specified by the request. If the port exists and is in use, it is hung up and unregistered first.

The port is then probed and if necessary the IRQ is autodetected If this fails an error is returned.

On success the port is ready to use and the line number is returned.

serial8250_unregister_port

LINUX

Kernel Hackers ManualApril 2016

Name

`serial8250_unregister_port` — remove a 16x50 serial port at runtime

Synopsis

```
void serial8250_unregister_port (int line);
```

Arguments

`line`

serial line number

Description

Remove one serial port. This may not be called from interrupt context. We hand the port back to the our control.

Chapter 7. Frame Buffer Library

The frame buffer drivers depend heavily on four data structures. These structures are declared in include/linux/fb.h. They are fb_info, fb_var_screeninfo, fb_fix_screeninfo and fb_monospecs. The last three can be made available to and from userland.

fb_info defines the current state of a particular video card. Inside fb_info, there exists a fb_ops structure which is a collection of needed functions to make fbdev and fbcon work. fb_info is only visible to the kernel.

fb_var_screeninfo is used to describe the features of a video card that are user defined. With fb_var_screeninfo, things such as depth and the resolution may be defined.

The next structure is fb_fix_screeninfo. This defines the properties of a card that are created when a mode is set and can't be changed otherwise. A good example of this is the start of the frame buffer memory. This "locks" the address of the frame buffer memory, so that it cannot be changed or moved.

The last structure is fb_monospecs. In the old API, there was little importance for fb_monospecs. This allowed for forbidden things such as setting a mode of 800x600 on a fix frequency monitor. With the new API, fb_monospecs prevents such things, and if used correctly, can prevent a monitor from being cooked. fb_monospecs will not be useful until kernels 2.5.x.

7.1. Frame Buffer Memory

register_framebuffer

LINUX

Kernel Hackers ManualApril 2016

Name

`register_framebuffer` — registers a frame buffer device

Synopsis

```
int register_framebuffer (struct fb_info * fb_info);
```

Arguments

fb_info

frame buffer info structure

Description

Registers a frame buffer device *fb_info*.

Returns negative errno on error, or zero for success.

unregister_framebuffer

LINUX

Kernel Hackers ManualApril 2016

Name

unregister_framebuffer — releases a frame buffer device

Synopsis

```
int unregister_framebuffer (struct fb_info * fb_info);
```

Arguments

fb_info

frame buffer info structure

Description

Unregisters a frame buffer device *fb_info*.

Returns negative errno on error, or zero for success.

This function will also notify the framebuffer console to release the driver.

This is meant to be called within a driver's `module_exit` function. If this is called outside `module_exit`, ensure that the driver implements `fb_open` and `fb_release` to check that no processes are using the device.

fb_set_suspend

LINUX

Kernel Hackers ManualApril 2016

Name

`fb_set_suspend` — low level driver signals suspend

Synopsis

```
void fb_set_suspend (struct fb_info * info, int state);
```

Arguments

info

framebuffer affected

state

0 = resuming, !=0 = suspending

Description

This is meant to be used by low level drivers to signal suspend/resume to the core & clients. It must be called with the console semaphore held

fb_get_options

LINUX

Kernel Hackers ManualApril 2016

Name

`fb_get_options` — get kernel boot parameters

Synopsis

```
int fb_get_options (char * name, char ** option);
```

Arguments

name

framebuffer name as it would appear in the boot parameter line (video=<name>:<options>)

option

the option will be stored here

NOTE

Needed to maintain backwards compatibility

7.2. Frame Buffer Colormap

fb_dealloc_cmap

LINUX

Kernel Hackers ManualApril 2016

Name

`fb_dealloc_cmap` — deallocate a colormap

Synopsis

```
void fb_dealloc_cmap (struct fb_cmap * cmap);
```

Arguments

cmap

frame buffer colormap structure

Description

Deallocates a colormap that was previously allocated with `fb_alloc_cmap`.

fb_copy_cmap

LINUX

Kernel Hackers ManualApril 2016

Name

`fb_copy_cmap` — copy a colormap

Synopsis

```
int fb_copy_cmap (const struct fb_cmap * from, struct fb_cmap
* to);
```

Arguments

`from`

frame buffer colormap structure

`to`

frame buffer colormap structure

Description

Copy contents of colormap from *`from`* to *`to`*.

fb_set_cmap

LINUX

Kernel Hackers ManualApril 2016

Name

`fb_set_cmap` — set the colormap

Synopsis

```
int fb_set_cmap (struct fb_cmap * cmap, struct fb_info *
info);
```

Arguments

cmap

frame buffer colormap structure

info

frame buffer info structure

Description

Sets the colormap *cmap* for a screen of device *info*.

Returns negative errno on error, or zero on success.

fb_default_cmap

LINUX

Name

`fb_default_cmap` — get default colormap

Synopsis

```
const struct fb_cmap * fb_default_cmap (int len);
```

Arguments

`len`

size of palette for a depth

Description

Gets the default colormap for a specific screen depth. *`len`* is the size of the palette for a particular screen depth.

Returns pointer to a frame buffer colormap structure.

fb_invert_cmaps

LINUX

Name

`fb_invert_cmaps` — invert all defaults colormaps

Synopsis

```
void fb_invert_cmaps ( void);
```

Arguments

void

no arguments

Description

Invert all default colormaps.

7.3. Frame Buffer Video Mode Database

fb_try_mode

LINUX

Kernel Hackers ManualApril 2016

Name

fb_try_mode — test a video mode

Synopsis

```
int fb_try_mode (struct fb_var_screeninfo * var, struct
fb_info * info, const struct fb_videomode * mode, unsigned int
```

bpp);

Arguments

var

frame buffer user defined part of display

info

frame buffer info structure

mode

frame buffer video mode structure

bpp

color depth in bits per pixel

Description

Tries a video mode to test it's validity for device *info*.

Returns 1 on success.

fb_delete_videomode

LINUX

Kernel Hackers ManualApril 2016

Name

fb_delete_videomode — removed videomode entry from modelist

Synopsis

```
void fb_delete_videomode (const struct fb_videomode * mode,
struct list_head * head);
```

Arguments

mode

videomode to remove

head

struct list_head of modelist

NOTES

Will remove all matching mode entries

fb_find_mode

LINUX

Kernel Hackers ManualApril 2016

Name

fb_find_mode — finds a valid video mode

Synopsis

```
int fb_find_mode (struct fb_var_screeninfo * var, struct
fb_info * info, const char * mode_option, const struct
```

```
fb_videomode * db, unsigned int dbsize, const struct
fb_videomode * default_mode, unsigned int default_bpp);
```

Arguments

var

frame buffer user defined part of display

info

frame buffer info structure

mode_option

string video mode to find

db

video mode database

dbsize

size of *db*

default_mode

default video mode to fall back to

default_bpp

default color depth in bits per pixel

Description

Finds a suitable video mode, starting with the specified mode in *mode_option* with fallback to *default_mode*. If *default_mode* fails, all modes in the video mode database will be tried.

Valid mode specifiers for *mode_option*:

<xres>x<yres>[M][R][-<bpp>][@<refresh>][i][m] or
<name>[-<bpp>][@<refresh>]

with <xres>, <yres>, <bpp> and <refresh> decimal numbers and <name> a string.

If 'M' is present after yres (and before refresh/bpp if present), the function will compute the timings using VESA(tm) Coordinated Video Timings (CVT). If 'R' is present after 'M', will compute with reduced blanking (for flatpanels). If 'i' is present, compute interlaced mode. If 'm' is present, add margins equal to 1.8% of xres rounded down to 8 pixels, and 1.8% of yres. The char 'i' and 'm' must be after 'M' and 'R'. Example:

1024x768MR-8*60m* - Reduced blank with margins at 60Hz.

NOTE

The passed struct *var* is _not_ cleared! This allows you to supply values for e.g. the grayscale and accel_flags fields.

Returns zero for failure, 1 if using specified *mode_option*, 2 if using specified *mode_option* with an ignored refresh rate, 3 if default mode is used, 4 if fall back to any valid mode.

fb_var_to_videomode

LINUX

Kernel Hackers ManualApril 2016

Name

`fb_var_to_videomode` — convert fb_var_screeninfo to fb_videomode

Synopsis

```
void fb_var_to_videomode (struct fb_videomode * mode, const
struct fb_var_screeninfo * var);
```

Arguments

mode

pointer to struct fb_videomode

var

pointer to struct fb_var_screeninfo

fb_videomode_to_var

LINUX

Kernel Hackers ManualApril 2016

Name

`fb_videomode_to_var` — convert fb_videomode to fb_var_screeninfo

Synopsis

```
void fb_videomode_to_var (struct fb_var_screeninfo * var,
const struct fb_videomode * mode);
```

Arguments

var

pointer to struct fb_var_screeninfo

mode

pointer to struct fb_videomode

fb_mode_is_equal

LINUX

Kernel Hackers ManualApril 2016

Name

`fb_mode_is_equal` — compare 2 videomodes

Synopsis

```
int fb_mode_is_equal (const struct fb_videomode * mode1, const
struct fb_videomode * mode2);
```

Arguments

mode1

first videomode

mode2

second videomode

RETURNS

1 if equal, 0 if not

fb_find_best_mode

LINUX

Name

`fb_find_best_mode` — find best matching videomode

Synopsis

```
const struct fb_videomode * fb_find_best_mode (const struct
fb_var_screeninfo * var, struct list_head * head);
```

Arguments

var

pointer to struct fb_var_screeninfo

head

pointer to struct list_head of modelist

RETURNS

struct fb_videomode, NULL if none found

IMPORTANT

This function assumes that all modelist entries in info->modelist are valid.

NOTES

Finds best matching videomode which has an equal or greater dimension than var->xres and var->yres. If more than 1 videomode is found, will return the videomode with the highest refresh rate

fb_find_nearest_mode

LINUX

Kernel Hackers ManualApril 2016

Name

`fb_find_nearest_mode` — find closest videomode

Synopsis

```
const struct fb_videomode * fb_find_nearest_mode (const struct
fb_videomode * mode, struct list_head * head);
```

Arguments

mode

pointer to struct fb_videomode

head

pointer to modelist

Description

Finds best matching videomode, smaller or greater in dimension. If more than 1 videomode is found, will return the videomode with the closest refresh rate.

fb_match_mode

LINUX

Name

`fb_match_mode` — find a videomode which exactly matches the timings in var

Synopsis

```
const struct fb_videomode * fb_match_mode (const struct
fb_var_screeninfo * var, struct list_head * head);
```

Arguments

var

pointer to struct fb_var_screeninfo

head

pointer to struct list_head of modelist

RETURNS

struct fb_videomode, NULL if none found

fb_add_videomode

LINUX

Name

`fb_add_videomode` — adds videomode entry to modelist

Synopsis

```
int fb_add_videomode (const struct fb_videomode * mode, struct
list_head * head);
```

Arguments

mode

videomode to add

head

struct list_head of modelist

NOTES

Will only add unmatched mode entries

fb_destroy_modelist

LINUX

Kernel Hackers ManualApril 2016

Name

`fb_destroy_modelist` — destroy modelist

Synopsis

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

Arguments

head

struct list_head of modelist

fb_videomode_to_modelist

LINUX

Kernel Hackers ManualApril 2016

Name

`fb_videomode_to_modelist` — convert mode array to mode list

Synopsis

```
void fb_videomode_to_modelist (const struct fb_videomode *
modedb, int num, struct list_head * head);
```

Arguments

modedb

array of struct fb_videomode

num

number of entries in array

head

struct list_head of modelist

7.4. Frame Buffer Macintosh Video Mode Database

mac_vmode_to_var

LINUX

Kernel Hackers ManualApril 2016

Name

`mac_vmode_to_var` — converts vmode/cmode pair to var structure

Synopsis

```
int mac_vmode_to_var (int vmode, int cmode, struct
fb_var_screeninfo * var);
```

Arguments

vmode

MacOS video mode

cmode

MacOS color mode

var

frame buffer video mode structure

Description

Converts a MacOS vmode/cmode pair to a frame buffer video mode structure.

Returns negative errno on error, or zero for success.

mac_map_monitor_sense

LINUX

Kernel Hackers ManualApril 2016

Name

`mac_map_monitor_sense` — Convert monitor sense to vmode

Synopsis

```
int mac_map_monitor_sense (int sense);
```

Arguments

sense

Macintosh monitor sense number

Description

Converts a Macintosh monitor sense number to a MacOS vmode number.

Returns MacOS vmode video mode number.

mac_find_mode

LINUX

Name

`mac_find_mode` — find a video mode

Synopsis

```
int mac_find_mode (struct fb_var_screeninfo * var, struct
fb_info * info, const char * mode_option, unsigned int
default_bpp);
```

Arguments

`var`

frame buffer user defined part of display

`info`

frame buffer info structure

`mode_option`

video mode name (see mac_modedb[])

`default_bpp`

default color depth in bits per pixel

Description

Finds a suitable video mode. Tries to set mode specified by *`mode_option`*. If the name of the wanted mode begins with 'mac', the Mac video mode database will be used, otherwise it will fall back to the standard video mode database.

Note

Function marked as __init and can only be used during system boot.

Returns error code from fb_find_mode (see fb_find_mode function).

7.5. Frame Buffer Fonts

Refer to the file drivers/video/console/fonts.c for more information.

Chapter 8. Input Subsystem

8.1. Input core

struct input_absinfo

LINUX

Kernel Hackers ManualApril 2016

Name

`struct input_absinfo` — used by EVIOCGABS/EVIOCSABS ioctls

Synopsis

```
struct input_absinfo {
  __s32 value;
  __s32 minimum;
  __s32 maximum;
  __s32 fuzz;
  __s32 flat;
  __s32 resolution;
};
```

Members

value

latest reported value for the axis.

minimum

specifies minimum value for the axis.

maximum

specifies maximum value for the axis.

fuzz

specifies fuzz value that is used to filter noise from the event stream.

flat

values that are within this value will be discarded by joydev interface and reported as 0 instead.

resolution

specifies resolution for the values reported for the axis.

Description

Note that input core does not clamp reported values to the [minimum, maximum] limits, such task is left to userspace.

Resolution for main axes (ABS_X, ABS_Y, ABS_Z) is reported in units per millimeter (units/mm), resolution for rotational axes (ABS_RX, ABS_RY, ABS_RZ) is reported in units per radian.

struct input_keymap_entry

LINUX

Kernel Hackers ManualApril 2016

Name

`struct input_keymap_entry` — used by EVIOCGKEYCODE/EVIOCSKEYCODE ioctls

Synopsis

```
struct input_keymap_entry {
#define INPUT_KEYMAP_BY_INDEX (1 << 0)
  __u8 flags;
  __u8 len;
  __u16 index;
  __u32 keycode;
```

```
  __u8 scancode[32];
};
```

Members

flags

allows to specify how kernel should handle the request. For example, setting INPUT_KEYMAP_BY_INDEX flag indicates that kernel should perform lookup in keymap by *index* instead of *scancode*

len

length of the scancode that resides in *scancode* buffer.

index

index in the keymap, may be used instead of scancode

keycode

key code assigned to this scancode

scancode[32]

scancode represented in machine-endian form.

Description

The structure is used to retrieve and modify keymap data. Users have option of performing lookup either by *scancode* itself or by *index* in keymap entry. EVIOCGKEYCODE will also return scancode or index (depending on which element was used to perform lookup).

struct ff_replay

LINUX

Name

`struct ff_replay` — defines scheduling of the force-feedback effect

Synopsis

```
struct ff_replay {
  __u16 length;
  __u16 delay;
};
```

Members

length

duration of the effect

delay

delay before effect should start playing

struct ff_trigger

LINUX

Name

`struct ff_trigger` — defines what triggers the force-feedback effect

Synopsis

```
struct ff_trigger {
  __u16 button;
```

```
  __u16 interval;
};
```

Members

button

number of the button triggering the effect

interval

controls how soon the effect can be re-triggered

struct ff_envelope

LINUX

Kernel Hackers ManualApril 2016

Name

`struct ff_envelope` — generic force-feedback effect envelope

Synopsis

```
struct ff_envelope {
  __u16 attack_length;
  __u16 attack_level;
  __u16 fade_length;
  __u16 fade_level;
};
```

Members

attack_length

duration of the attack (ms)

attack_level

level at the beginning of the attack

fade_length

duration of fade (ms)

fade_level

level at the end of fade

Description

The *attack_level* and *fade_level* are absolute values; when applying envelope force-feedback core will convert to positive/negative value based on polarity of the default level of the effect. Valid range for the attack and fade levels is 0x0000 - 0x7fff

struct ff_constant_effect

LINUX

Kernel Hackers ManualApril 2016

Name

`struct ff_constant_effect` — defines parameters of a constant force-feedback effect

Synopsis

```
struct ff_constant_effect {
  __s16 level;
  struct ff_envelope envelope;
};
```

Members

level

strength of the effect; may be negative

envelope

envelope data

struct ff_ramp_effect

LINUX

Kernel Hackers ManualApril 2016

Name

`struct ff_ramp_effect` — defines parameters of a ramp force-feedback effect

Synopsis

```
struct ff_ramp_effect {
  __s16 start_level;
  __s16 end_level;
  struct ff_envelope envelope;
};
```

Members

start_level

beginning strength of the effect; may be negative

end_level

final strength of the effect; may be negative

envelope

envelope data

struct ff_condition_effect

LINUX

Kernel Hackers ManualApril 2016

Name

`struct ff_condition_effect` — defines a spring or friction force-feedback effect

Synopsis

```
struct ff_condition_effect {
  __u16 right_saturation;
  __u16 left_saturation;
  __s16 right_coeff;
  __s16 left_coeff;
  __u16 deadband;
  __s16 center;
};
```

Members

right_saturation

maximum level when joystick moved all way to the right

left_saturation

same for the left side

right_coeff

controls how fast the force grows when the joystick moves to the right

left_coeff

same for the left side

deadband

size of the dead zone, where no force is produced

center

position of the dead zone

struct ff_periodic_effect

LINUX

Kernel Hackers ManualApril 2016

Name

`struct ff_periodic_effect` — defines parameters of a periodic force-feedback effect

Synopsis

```
struct ff_periodic_effect {
  __u16 waveform;
  __u16 period;
  __s16 magnitude;
  __s16 offset;
  __u16 phase;
  struct ff_envelope envelope;
  __u32 custom_len;
  __s16 __user * custom_data;
};
```

Members

waveform

kind of the effect (wave)

period

period of the wave (ms)

magnitude

peak value

offset

mean value of the wave (roughly)

phase

'horizontal' shift

envelope

envelope data

custom_len

number of samples (FF_CUSTOM only)

custom_data

buffer of samples (FF_CUSTOM only)

Description

Known waveforms - FF_SQUARE, FF_TRIANGLE, FF_SINE, FF_SAW_UP, FF_SAW_DOWN, FF_CUSTOM. The exact syntax FF_CUSTOM is undefined for the time being as no driver supports it yet.

Note

the data pointed by custom_data is copied by the driver. You can therefore dispose of the memory after the upload/update.

struct ff_rumble_effect

LINUX

Kernel Hackers ManualApril 2016

Name

`struct ff_rumble_effect` — defines parameters of a periodic force-feedback effect

Synopsis

```
struct ff_rumble_effect {
  __u16 strong_magnitude;
  __u16 weak_magnitude;
};
```

Members

strong_magnitude

magnitude of the heavy motor

weak_magnitude

magnitude of the light one

Description

Some rumble pads have two motors of different weight. Strong_magnitude represents the magnitude of the vibration generated by the heavy one.

struct ff_effect

LINUX

Name

`struct ff_effect` — defines force feedback effect

Synopsis

```
struct ff_effect {
  __u16 type;
  __s16 id;
  __u16 direction;
  struct ff_trigger trigger;
  struct ff_replay replay;
  union u;
};
```

Members

type

type of the effect (FF_CONSTANT, FF_PERIODIC, FF_RAMP, FF_SPRING, FF_FRICTION, FF_DAMPER, FF_RUMBLE, FF_INERTIA, or FF_CUSTOM)

id

an unique id assigned to an effect

direction

direction of the effect

trigger

trigger conditions (struct ff_trigger)

replay

scheduling of the effect (struct ff_replay)

u

effect-specific structure (one of ff_constant_effect, ff_ramp_effect, ff_periodic_effect, ff_condition_effect, ff_rumble_effect) further defining effect parameters

Description

This structure is sent through ioctl from the application to the driver. To create a new effect application should set its `id` to -1; the kernel will return assigned `id` which can later be used to update or delete this effect.

Direction of the effect is encoded as follows

0 deg -> 0x0000 (down) 90 deg -> 0x4000 (left) 180 deg -> 0x8000 (up) 270 deg -> 0xC000 (right)

struct input_dev

LINUX

Kernel Hackers ManualApril 2016

Name

`struct input_dev` — represents an input device

Synopsis

```
struct input_dev {
  const char * name;
  const char * phys;
  const char * uniq;
  struct input_id id;
  unsigned long propbit[BITS_TO_LONGS(INPUT_PROP_CNT)];
  unsigned long evbit[BITS_TO_LONGS(EV_CNT)];
  unsigned long keybit[BITS_TO_LONGS(KEY_CNT)];
  unsigned long relbit[BITS_TO_LONGS(REL_CNT)];
```

```
  unsigned long absbit[BITS_TO_LONGS(ABS_CNT)];
  unsigned long mscbit[BITS_TO_LONGS(MSC_CNT)];
  unsigned long ledbit[BITS_TO_LONGS(LED_CNT)];
  unsigned long sndbit[BITS_TO_LONGS(SND_CNT)];
  unsigned long ffbit[BITS_TO_LONGS(FF_CNT)];
  unsigned long swbit[BITS_TO_LONGS(SW_CNT)];
  unsigned int hint_events_per_packet;
  unsigned int keycodemax;
  unsigned int keycodesize;
  void * keycode;
  int (* setkeycode) (struct input_dev *dev,const struct input_keymap_ent
  int (* getkeycode) (struct input_dev *dev,struct input_keymap_entry *ke
  struct ff_device * ff;
  unsigned int repeat_key;
  struct timer_list timer;
  int rep[REP_CNT];
  struct input_mt_slot * mt;
  int mtsize;
  int slot;
  int trkid;
  struct input_absinfo * absinfo;
  unsigned long key[BITS_TO_LONGS(KEY_CNT)];
  unsigned long led[BITS_TO_LONGS(LED_CNT)];
  unsigned long snd[BITS_TO_LONGS(SND_CNT)];
  unsigned long sw[BITS_TO_LONGS(SW_CNT)];
  int (* open) (struct input_dev *dev);
  void (* close) (struct input_dev *dev);
  int (* flush) (struct input_dev *dev, struct file *file);
  int (* event) (struct input_dev *dev, unsigned int type, unsigned int co
  struct input_handle __rcu * grab;
  spinlock_t event_lock;
  struct mutex mutex;
  unsigned int users;
  bool going_away;
  bool sync;
  struct device dev;
  struct list_head h_list;
  struct list_head node;
};
```

Members

name

name of the device

phys

: physical path to the device in the system hierarchy

uniq

: unique identification code for the device (if device has it)

id

: id of the device (struct input_id)

propbit[BITS_TO_LONGS(INPUT_PROP_CNT)]

: bitmap of device properties and quirks

evbit[BITS_TO_LONGS(EV_CNT)]

: bitmap of types of events supported by the device (EV_KEY, EV_REL, etc.)

keybit[BITS_TO_LONGS(KEY_CNT)]

: bitmap of keys/buttons this device has

relbit[BITS_TO_LONGS(REL_CNT)]

: bitmap of relative axes for the device

absbit[BITS_TO_LONGS(ABS_CNT)]

: bitmap of absolute axes for the device

mscbit[BITS_TO_LONGS(MSC_CNT)]

: bitmap of miscellaneous events supported by the device

ledbit[BITS_TO_LONGS(LED_CNT)]

: bitmap of leds present on the device

sndbit[BITS_TO_LONGS(SND_CNT)]

: bitmap of sound effects supported by the device

ffbit[BITS_TO_LONGS(FF_CNT)]

: bitmap of force feedback effects supported by the device

swbit[BITS_TO_LONGS(SW_CNT)]

: bitmap of switches present on the device

hint_events_per_packet

: average number of events generated by the device in a packet (between EV_SYN/SYN_REPORT events). Used by event handlers to estimate size of the buffer needed to hold events.

keycodemax

: size of keycode table

keycodesize

: size of elements in keycode table

keycode

: map of scancodes to keycodes for this device

setkeycode

: optional method to alter current keymap, used to implement sparse keymaps. If not supplied default mechanism will be used. The method is being called while holding event_lock and thus must not sleep

getkeycode

: optional legacy method to retrieve current keymap.

ff

: force feedback structure associated with the device if device supports force feedback effects

repeat_key

: stores key code of the last key pressed; used to implement software autorepeat

timer

: timer for software autorepeat

rep[REP_CNT]

: current values for autorepeat parameters (delay, rate)

mt

: pointer to array of struct input_mt_slot holding current values of tracked contacts

mtsize

: number of MT slots the device uses

slot

MT slot currently being transmitted

trkid

stores MT tracking ID for the current contact

absinfo

array of struct input_absinfo elements holding information about absolute axes (current value, min, max, flat, fuzz, resolution)

key[BITS_TO_LONGS(KEY_CNT)]

reflects current state of device's keys/buttons

led[BITS_TO_LONGS(LED_CNT)]

reflects current state of device's LEDs

snd[BITS_TO_LONGS(SND_CNT)]

reflects current state of sound effects

sw[BITS_TO_LONGS(SW_CNT)]

reflects current state of device's switches

open

this method is called when the very first user calls `input_open_device`. The driver must prepare the device to start generating events (start polling thread, request an IRQ, submit URB, etc.)

close

this method is called when the very last user calls `input_close_device`.

flush

purges the device. Most commonly used to get rid of force feedback effects loaded into the device when disconnecting from it

event

event handler for events sent _to_ the device, like EV_LED or EV_SND. The device is expected to carry out the requested action (turn on a LED, play sound, etc.) The call is protected by *`event_lock`* and must not sleep

grab

input handle that currently has the device grabbed (via EVIOCGRAB ioctl). When a handle grabs a device it becomes sole recipient for all input events coming from the device

event_lock

this spinlock is is taken when input core receives and processes a new event for the device (in `input_event`). Code that accesses and/or modifies parameters of a device (such as keymap or absmin, absmax, absfuzz, etc.) after device has been registered with input core must take this lock.

mutex

serializes calls to `open`, `close` and `flush` methods

users

stores number of users (input handlers) that opened this device. It is used by `input_open_device` and `input_close_device` to make sure that dev->`open` is only called when the first user opens device and dev->`close` is called when the very last user closes the device

going_away

marks devices that are in a middle of unregistering and causes input_open_device*() fail with -ENODEV.

sync

set to `true` when there were no new events since last EV_SYN

dev

driver model's view of this device

h_list

list of input handles associated with the device. When accessing the list dev->mutex must be held

node

used to place the device onto input_dev_list

struct input_handler

LINUX

Kernel Hackers ManualApril 2016

Name

`struct input_handler` — implements one of interfaces for input devices

Synopsis

```
struct input_handler {
  void * private;
  void (* event) (struct input_handle *handle, unsigned int type, unsigned
  bool (* filter) (struct input_handle *handle, unsigned int type, unsigne
  bool (* match) (struct input_handler *handler, struct input_dev *dev);
  int (* connect) (struct input_handler *handler, struct input_dev *dev, c
  void (* disconnect) (struct input_handle *handle);
  void (* start) (struct input_handle *handle);
  const struct file_operations * fops;
  int minor;
  const char * name;
  const struct input_device_id * id_table;
  struct list_head h_list;
  struct list_head node;
};
```

Members

private

driver-specific data

event

event handler. This method is being called by input core with interrupts disabled and dev->event_lock spinlock held and so it may not sleep

filter

similar to *event*; separates normal event handlers from “filters”.

match

called after comparing device's id with handler's id_table to perform fine-grained matching between device and handler

connect

called when attaching a handler to an input device

disconnect

disconnects a handler from input device

start

starts handler for given handle. This function is called by input core right after `connect` method and also when a process that "grabbed" a device releases it

fops

file operations this driver implements

minor

beginning of range of 32 minors for devices this driver can provide

name

name of the handler, to be shown in /proc/bus/input/handlers

id_table

pointer to a table of input_device_ids this driver can handle

h_list

list of input handles associated with the handler

node

for placing the driver onto input_handler_list

Description

Input handlers attach to input devices and create input handles. There are likely several handlers attached to any given input device at the same time. All of them will get their copy of input event generated by the device.

The very same structure is used to implement input filters. Input core allows filters to run first and will not pass event to regular handlers if any of the filters indicate that the event should be filtered (by returning `true` from their `filter` method).

Note that input core serializes calls to `connect` and `disconnect` methods.

struct input_handle

LINUX

Kernel Hackers ManualApril 2016

Name

`struct input_handle` — links input device with an input handler

Synopsis

```
struct input_handle {
  void * private;
  int open;
  const char * name;
  struct input_dev * dev;
  struct input_handler * handler;
  struct list_head d_node;
  struct list_head h_node;
};
```

Members

private

handler-specific data

open

counter showing whether the handle is 'open', i.e. should deliver events from its device

name

name given to the handle by handler that created it

dev

input device the handle is attached to

handler

handler that works with the device through this handle

d_node

used to put the handle on device's list of attached handles

h_node

used to put the handle on handler's list of handles from which it gets events

input_set_events_per_packet

LINUX

Kernel Hackers ManualApril 2016

Name

`input_set_events_per_packet` — tell handlers about the driver event rate

Synopsis

```
void input_set_events_per_packet (struct input_dev * dev, int
n_events);
```

Arguments

dev

the input device used by the driver

n_events

the average number of events between calls to `input_sync`

Description

If the event rate sent from a device is unusually large, use this function to set the expected event rate. This will allow handlers to set up an appropriate buffer size for the event stream, in order to minimize information loss.

struct ff_device

LINUX

Kernel Hackers ManualApril 2016

Name

`struct ff_device` — force-feedback part of an input device

Synopsis

```
struct ff_device {
  int (* upload) (struct input_dev *dev, struct ff_effect *effect,struct
  int (* erase) (struct input_dev *dev, int effect_id);
  int (* playback) (struct input_dev *dev, int effect_id, int value);
  void (* set_gain) (struct input_dev *dev, u16 gain);
  void (* set_autocenter) (struct input_dev *dev, u16 magnitude);
  void (* destroy) (struct ff_device *);
  void * private;
  unsigned long ffbit[BITS_TO_LONGS(FF_CNT)];
  struct mutex mutex;
  int max_effects;
  struct ff_effect * effects;
  struct file * effect_owners[];
};
```

Members

upload

Called to upload an new effect into device

erase

Called to erase an effect from device

playback

Called to request device to start playing specified effect

set_gain

Called to set specified gain

set_autocenter

Called to auto-center device

destroy

called by input core when parent input device is being destroyed

private

driver-specific data, will be freed automatically

ffbit[BITS_TO_LONGS(FF_CNT)]

bitmap of force feedback capabilities truly supported by device (not emulated like ones in input_dev->ffbit)

mutex

mutex for serializing access to the device

max_effects

maximum number of effects supported by device

effects

pointer to an array of effects currently loaded into device

effect_owners[]

array of effect owners; when file handle owning an effect gets closed the effect is automatically erased

Description

Every force-feedback device must implement `upload` and `playback` methods; `erase` is optional. `set_gain` and `set_autocenter` need only be implemented if driver sets up FF_GAIN and FF_AUTOCENTER bits.

Note that `playback`, `set_gain` and `set_autocenter` are called with dev->event_lock spinlock held and interrupts off and thus may not sleep.

input_event

LINUX

Kernel Hackers ManualApril 2016

Name

`input_event` — report new input event

Synopsis

```
void input_event (struct input_dev * dev, unsigned int type,
unsigned int code, int value);
```

Arguments

dev

device that generated the event

type

type of the event

code

event code

value

value of the event

Description

This function should be used by drivers implementing various input devices to report input events. See also `input_inject_event`.

NOTE

`input_event` may be safely used right after input device was allocated with `input_allocate_device`, even before it is registered with `input_register_device`, but the event will not reach any of the input handlers. Such early invocation of `input_event` may be used to 'seed' initial state of a switch or initial position of absolute axis, etc.

input_inject_event

LINUX

Kernel Hackers ManualApril 2016

Name

`input_inject_event` — send input event from input handler

Synopsis

```
void input_inject_event (struct input_handle * handle,
unsigned int type, unsigned int code, int value);
```

Arguments

handle

input handle to send event through

type

type of the event

code

event code

value

value of the event

Description

Similar to `input_event` but will ignore event if device is “grabbed” and handle injecting event is not the one that owns the device.

input_alloc_absinfo

LINUX

Kernel Hackers ManualApril 2016

Name

`input_alloc_absinfo` — allocates array of input_absinfo structs

Synopsis

```
void input_alloc_absinfo (struct input_dev * dev);
```

Arguments

dev

the input device emitting absolute events

Description

If the absinfo struct the caller asked for is already allocated, this functions will not do anything.

input_grab_device

LINUX

Kernel Hackers ManualApril 2016

Name

input_grab_device — grabs device for exclusive use

Synopsis

```
int input_grab_device (struct input_handle * handle);
```

Arguments

handle

input handle that wants to own the device

Description

When a device is grabbed by an input handle all events generated by the device are delivered only to this handle. Also events injected by other input handles are ignored while device is grabbed.

input_release_device

LINUX

Kernel Hackers ManualApril 2016

Name

`input_release_device` — release previously grabbed device

Synopsis

```
void input_release_device (struct input_handle * handle);
```

Arguments

`handle`

input handle that owns the device

Description

Releases previously grabbed device so that other input handles can start receiving input events. Upon release all handlers attached to the device have their `start` method called so they have a change to synchronize device state with the rest of the system.

input_open_device

LINUX

Kernel Hackers ManualApril 2016

Name

`input_open_device` — open input device

Synopsis

```
int input_open_device (struct input_handle * handle);
```

Arguments

`handle`

handle through which device is being accessed

Description

This function should be called by input handlers when they want to start receive events from given input device.

input_close_device

LINUX

Name

`input_close_device` — close input device

Synopsis

```
void input_close_device (struct input_handle * handle);
```

Arguments

`handle`

handle through which device is being accessed

Description

This function should be called by input handlers when they want to stop receive events from given input device.

input_scancode_to_scalar

LINUX

Name

`input_scancode_to_scalar` — converts scancode in struct input_keymap_entry

Synopsis

```
int input_scancode_to_scalar (const struct input_keymap_entry
* ke, unsigned int * scancode);
```

Arguments

ke

keymap entry containing scancode to be converted.

scancode

pointer to the location where converted scancode should be stored.

Description

This function is used to convert scancode stored in struct keymap_entry into scalar form understood by legacy keymap handling methods. These methods expect scancodes to be represented as 'unsigned int'.

input_get_keycode

LINUX

Kernel Hackers ManualApril 2016

Name

`input_get_keycode` — retrieve keycode currently mapped to a given scancode

Synopsis

```
int input_get_keycode (struct input_dev * dev, struct
input_keymap_entry * ke);
```

Arguments

dev

input device which keymap is being queried

ke

keymap entry

Description

This function should be called by anyone interested in retrieving current keymap. Presently evdev handlers use it.

input_set_keycode

LINUX

Kernel Hackers ManualApril 2016

Name

`input_set_keycode` — attribute a keycode to a given scancode

Synopsis

```
int input_set_keycode (struct input_dev * dev, const struct
input_keymap_entry * ke);
```

Arguments

dev

input device which keymap is being updated

ke

new keymap entry

Description

This function should be called by anyone needing to update current keymap. Presently keyboard and evdev handlers use it.

input_reset_device

LINUX

Kernel Hackers ManualApril 2016

Name

`input_reset_device` — reset/restore the state of input device

Synopsis

```
void input_reset_device (struct input_dev * dev);
```

Arguments

dev

input device whose state needs to be reset

Description

This function tries to reset the state of an opened input device and bring internal state and state if the hardware in sync with each other. We mark all keys as released, restore LED state, repeat rate, etc.

input_allocate_device

LINUX

Kernel Hackers ManualApril 2016

Name

`input_allocate_device` — allocate memory for new input device

Synopsis

```
struct input_dev * input_allocate_device ( void);
```

Arguments

void

no arguments

Description

Returns prepared struct input_dev or NULL.

NOTE

Use `input_free_device` to free devices that have not been registered; `input_unregister_device` should be used for already registered devices.

input_free_device

LINUX

Kernel Hackers ManualApril 2016

Name

`input_free_device` — free memory occupied by input_dev structure

Synopsis

```
void input_free_device (struct input_dev * dev);
```

Arguments

dev

input device to free

Description

This function should only be used if `input_register_device` was not called yet or if it failed. Once device was registered use `input_unregister_device` and memory will be freed once last reference to the device is dropped.

Device should be allocated by `input_allocate_device`.

NOTE

If there are references to the input device then memory will not be freed until last reference is dropped.

input_set_capability

LINUX

Kernel Hackers ManualApril 2016

Name

`input_set_capability` — mark device as capable of a certain event

Synopsis

```
void input_set_capability (struct input_dev * dev, unsigned
int type, unsigned int code);
```

Arguments

dev

device that is capable of emitting or accepting event

type

type of the event (EV_KEY, EV_REL, etc...)

code

event code

Description

In addition to setting up corresponding bit in appropriate capability bitmap the function also adjusts dev->evbit.

input_register_device

LINUX

Kernel Hackers ManualApril 2016

Name

`input_register_device` — register device with input core

Synopsis

```
int input_register_device (struct input_dev * dev);
```

Arguments

dev

device to be registered

Description

This function registers device with input core. The device must be allocated with `input_allocate_device` and all it's capabilities set up before registering. If function fails the device must be freed with `input_free_device`. Once device has been successfully registered it can be unregistered with `input_unregister_device`; `input_free_device` should not be called in this case.

input_unregister_device

LINUX

Kernel Hackers ManualApril 2016

Name

`input_unregister_device` — unregister previously registered device

Synopsis

```
void input_unregister_device (struct input_dev * dev);
```

Arguments

dev

device to be unregistered

Description

This function unregisters an input device. Once device is unregistered the caller should not try to access it as it may get freed at any moment.

input_register_handler

LINUX

Kernel Hackers ManualApril 2016

Name

`input_register_handler` — register a new input handler

Synopsis

```
int input_register_handler (struct input_handler * handler);
```

Arguments

handler

handler to be registered

Description

This function registers a new input handler (interface) for input devices in the system and attaches it to all input devices that are compatible with the handler.

input_unregister_handler

LINUX

Name

`input_unregister_handler` — unregisters an input handler

Synopsis

```
void input_unregister_handler (struct input_handler *
handler);
```

Arguments

handler

handler to be unregistered

Description

This function disconnects a handler from its input devices and removes it from lists of known handlers.

input_handler_for_each_handle

LINUX

Name

`input_handler_for_each_handle` — handle iterator

Synopsis

```
int input_handler_for_each_handle (struct input_handler *
handler, void * data, int (*fn) (struct input_handle *, void
*));
```

Arguments

handler

input handler to iterate

data

data for the callback

fn

function to be called for each handle

Description

Iterate over *bus*'s list of devices, and call *fn* for each, passing it *data* and stop when *fn* returns a non-zero value. The function is using RCU to traverse the list and therefore may be usind in atonic contexts. The *fn* callback is invoked from RCU critical section and thus must not sleep.

input_register_handle

LINUX

Kernel Hackers ManualApril 2016

Name

input_register_handle — register a new input handle

Synopsis

```
int input_register_handle (struct input_handle * handle);
```

Arguments

handle

handle to register

Description

This function puts a new input handle onto device's and handler's lists so that events can flow through it once it is opened using `input_open_device`.

This function is supposed to be called from handler's `connect` method.

input_unregister_handle

LINUX

Kernel Hackers ManualApril 2016

Name

`input_unregister_handle` — unregister an input handle

Synopsis

```
void input_unregister_handle (struct input_handle * handle);
```

Arguments

handle

handle to unregister

Description

This function removes input handle from device's and handler's lists.

This function is supposed to be called from handler's `disconnect` method.

input_ff_upload

LINUX

Kernel Hackers ManualApril 2016

Name

`input_ff_upload` — upload effect into force-feedback device

Synopsis

```
int input_ff_upload (struct input_dev * dev, struct ff_effect
* effect, struct file * file);
```

Arguments

dev

input device

effect

effect to be uploaded

file

owner of the effect

input_ff_erase

LINUX

Kernel Hackers ManualApril 2016

Name

`input_ff_erase` — erase a force-feedback effect from device

Synopsis

```
int input_ff_erase (struct input_dev * dev, int effect_id,
struct file * file);
```

Arguments

dev

input device to erase effect from

effect_id

id of the ffect to be erased

file

purported owner of the request

Description

This function erases a force-feedback effect from specified device. The effect will only be erased if it was uploaded through the same file handle that is requesting erase.

input_ff_event

LINUX

Kernel Hackers ManualApril 2016

Name

`input_ff_event` — generic handler for force-feedback events

Synopsis

```
int input_ff_event (struct input_dev * dev, unsigned int type,
unsigned int code, int value);
```

Arguments

dev

input device to send the effect to

type

event type (anything but EV_FF is ignored)

code

event code

value

event value

input_ff_create

LINUX

Kernel Hackers ManualApril 2016

Name

`input_ff_create` — create force-feedback device

Synopsis

```
int input_ff_create (struct input_dev * dev, int max_effects);
```

Arguments

dev

input device supporting force-feedback

max_effects

maximum number of effects supported by the device

Description

This function allocates all necessary memory for a force feedback portion of an input device and installs all default handlers. *dev*->ffbit should be already set up before calling this function. Once ff device is created you need to setup its upload, erase, playback and other handlers before registering input device

input_ff_destroy

LINUX

Kernel Hackers ManualApril 2016

Name

`input_ff_destroy` — frees force feedback portion of input device

Synopsis

```
void input_ff_destroy (struct input_dev * dev);
```

Arguments

dev

input device supporting force feedback

Description

This function is only needed in error path as input core will automatically free force feedback structures when device is destroyed.

input_ff_create_memless

LINUX

Name

`input_ff_create_memless` — create memoryless force-feedback device

Synopsis

```
int input_ff_create_memless (struct input_dev * dev, void *
data, int (*play_effect) (struct input_dev *, void *, struct
ff_effect *));
```

Arguments

dev

input device supporting force-feedback

data

driver-specific data to be passed into *play_effect*

play_effect

driver-specific method for playing FF effect

8.2. Multitouch Library

struct input_mt_slot

LINUX

Name

`struct input_mt_slot` — represents the state of an input MT slot

Synopsis

```
struct input_mt_slot {
  int abs[ABS_MT_LAST - ABS_MT_FIRST + 1];
};
```

Members

abs[ABS_MT_LAST - ABS_MT_FIRST + 1]

holds current values of ABS_MT axes for this slot

input_mt_init_slots

LINUX

Name

`input_mt_init_slots` — initialize MT input slots

Synopsis

```
int input_mt_init_slots (struct input_dev * dev, unsigned int
num_slots);
```

Arguments

dev

input device supporting MT events and finger tracking

num_slots

number of slots used by the device

Description

This function allocates all necessary memory for MT slot handling in the input device, prepares the ABS_MT_SLOT and ABS_MT_TRACKING_ID events for use and sets up appropriate buffers. May be called repeatedly. Returns -EINVAL if attempting to reinitialize with a different number of slots.

input_mt_destroy_slots

LINUX

Kernel Hackers ManualApril 2016

Name

`input_mt_destroy_slots` — frees the MT slots of the input device

Synopsis

```
void input_mt_destroy_slots (struct input_dev * dev);
```

Arguments

dev

input device with allocated MT slots

Description

This function is only needed in error path as the input core will automatically free the MT slots when the device is destroyed.

input_mt_report_slot_state

LINUX

Kernel Hackers ManualApril 2016

Name

`input_mt_report_slot_state` — report contact state

Synopsis

```
void input_mt_report_slot_state (struct input_dev * dev,
unsigned int tool_type, bool active);
```

Arguments

dev

input device with allocated MT slots

tool_type

the tool type to use in this slot

active

true if contact is active, false otherwise

Description

Reports a contact via ABS_MT_TRACKING_ID, and optionally ABS_MT_TOOL_TYPE. If active is true and the slot is currently inactive, or if the tool type is changed, a new tracking id is assigned to the slot. The tool type is only reported if the corresponding absbit field is set.

input_mt_report_finger_count

LINUX

Kernel Hackers ManualApril 2016

Name

`input_mt_report_finger_count` — report contact count

Synopsis

```
void input_mt_report_finger_count (struct input_dev * dev, int
count);
```

Arguments

dev

input device with allocated MT slots

count

the number of contacts

Description

Reports the contact count via BTN_TOOL_FINGER, BTN_TOOL_DOUBLETAP, BTN_TOOL_TRIPLETAP and BTN_TOOL_QUADTAP.

The input core ensures only the KEY events already setup for this device will produce output.

input_mt_report_pointer_emulation

LINUX

Kernel Hackers ManualApril 2016

Name

`input_mt_report_pointer_emulation` — common pointer emulation

Synopsis

```
void input_mt_report_pointer_emulation (struct input_dev *
dev, bool use_count);
```

Arguments

dev

input device with allocated MT slots

use_count

report number of active contacts as finger count

Description

Performs legacy pointer emulation via BTN_TOUCH, ABS_X, ABS_Y and ABS_PRESSURE. Touchpad finger count is emulated if use_count is true.

The input core ensures only the KEY and ABS axes already setup for this device will produce output.

8.3. Polled input devices

struct input_polled_dev

LINUX

Kernel Hackers ManualApril 2016

Name

`struct input_polled_dev` — simple polled input device

Synopsis

```
struct input_polled_dev {
  void * private;
  void (* open) (struct input_polled_dev *dev);
  void (* close) (struct input_polled_dev *dev);
  void (* poll) (struct input_polled_dev *dev);
  unsigned int poll_interval;
  unsigned int poll_interval_max;
  unsigned int poll_interval_min;
  struct input_dev * input;
};
```

Members

private

private driver data.

open

driver-supplied method that prepares device for polling (enabled the device and maybe flushes device state).

close

driver-supplied method that is called when device is no longer being polled. Used to put device into low power mode.

poll

driver-supplied method that polls the device and posts input events (mandatory).

poll_interval

specifies how often the `poll` method should be called. Defaults to 500 msec unless overridden when registering the device.

poll_interval_max

specifies upper bound for the poll interval. Defaults to the initial value of *`poll_interval`*.

poll_interval_min

specifies lower bound for the poll interval. Defaults to 0.

input

input device structure associated with the polled device. Must be properly initialized by the driver (id, name, phys, bits).

Description

Polled input device provides a skeleton for supporting simple input devices that do not raise interrupts but have to be periodically scanned or polled to detect changes in their state.

input_allocate_polled_device

LINUX

Kernel Hackers ManualApril 2016

Name

`input_allocate_polled_device` — allocate memory for polled device

Synopsis

```
struct input_polled_dev * input_allocate_polled_device (
void);
```

Arguments

void

no arguments

Description

The function allocates memory for a polled device and also for an input device associated with this polled device.

input_free_polled_device

LINUX

Name

`input_free_polled_device` — free memory allocated for polled device

Synopsis

```
void input_free_polled_device (struct input_polled_dev * dev);
```

Arguments

dev

device to free

Description

The function frees memory allocated for polling device and drops reference to the associated input device.

input_register_polled_device

LINUX

Name

`input_register_polled_device` — register polled device

Synopsis

```
int input_register_polled_device (struct input_polled_dev *
dev);
```

Arguments

dev

device to register

Description

The function registers previously initialized polled input device with input layer. The device should be allocated with call to `input_allocate_polled_device`. Callers should also set up `poll` method and set up capabilities (id, name, phys, bits) of the corresponding input_dev structure.

input_unregister_polled_device

LINUX

Kernel Hackers ManualApril 2016

Name

`input_unregister_polled_device` — unregister polled device

Synopsis

```
void input_unregister_polled_device (struct input_polled_dev *
dev);
```

Arguments

dev

device to unregister

Description

The function unregisters previously registered polled input device from input layer. Polling is stopped and device is ready to be freed with call to `input_free_polled_device`.

8.4. Matrix keyboars/keypads

struct matrix_keymap_data

LINUX

Kernel Hackers ManualApril 2016

Name

`struct matrix_keymap_data` — keymap for matrix keyboards

Synopsis

```
struct matrix_keymap_data {
  const uint32_t * keymap;
  unsigned int keymap_size;
};
```

Members

keymap

pointer to array of uint32 values encoded with `KEY` macro representing keymap

keymap_size

number of entries (initialized) in this keymap

Description

This structure is supposed to be used by platform code to supply keymaps to drivers that implement matrix-like keypads/keyboards.

struct matrix_keypad_platform_data

LINUX

Kernel Hackers ManualApril 2016

Name

`struct matrix_keypad_platform_data` — platform-dependent keypad data

Synopsis

```
struct matrix_keypad_platform_data {
  const struct matrix_keymap_data * keymap_data;
  const unsigned int * row_gpios;
  const unsigned int * col_gpios;
  unsigned int num_row_gpios;
  unsigned int num_col_gpios;
  unsigned int col_scan_delay_us;
  unsigned int debounce_ms;
  unsigned int clustered_irq;
  unsigned int clustered_irq_flags;
  bool active_low;
```

```
  bool wakeup;
  bool no_autorepeat;
};
```

Members

keymap_data

pointer to matrix_keymap_data

row_gpios

pointer to array of gpio numbers representing rows

col_gpios

pointer to array of gpio numbers reporesenting colums

num_row_gpios

actual number of row gpios used by device

num_col_gpios

actual number of col gpios used by device

col_scan_delay_us

delay, measured in microseconds, that is needed before we can keypad after activating column gpio

debounce_ms

debounce interval in milliseconds

clustered_irq

may be specified if interrupts of all row/column GPIOs are bundled to one single irq

clustered_irq_flags

flags that are needed for the clustered irq

active_low

gpio polarity

wakeup

controls whether the device should be set up as wakeup source

no_autorepeat

disable key autorepeat

Description

This structure represents platform-specific data that use used by matrix_keypad driver to perform proper initialization.

matrix_keypad_build_keymap

LINUX

Kernel Hackers ManualApril 2016

Name

`matrix_keypad_build_keymap` — convert platform keymap into matrix keymap

Synopsis

```
void matrix_keypad_build_keymap (const struct
matrix_keymap_data * keymap_data, unsigned int row_shift,
unsigned short * keymap, unsigned long * keybit);
```

Arguments

`keymap_data`

keymap supplied by the platform code

row_shift

number of bits to shift row value by to advance to the next line in the keymap

keymap

expanded version of keymap that is suitable for use by matrix keyboad driver

keybit

pointer to bitmap of keys supported by input device

Description

This function converts platform keymap (encoded with `KEY` macro) into an array of keycodes that is suitable for using in a standard matrix keyboard driver that uses row and col as indices.

8.5. Sparse keymap support

struct key_entry

LINUX

Kernel Hackers ManualApril 2016

Name

`struct key_entry` — keymap entry for use in sparse keymap

Synopsis

```
struct key_entry {
  int type;
  u32 code;
  union {unnamed_union};
};
```

Members

type

Type of the key entry (KE_KEY, KE_SW, KE_VSW, KE_END); drivers are allowed to extend the list with their own private definitions.

code

Device-specific data identifying the button/switch

{unnamed_union}

anonymous

Description

This structure defines an entry in a sparse keymap used by some input devices for which traditional table-based approach is not suitable.

sparse_keymap_entry_from_scancode

LINUX

Kernel Hackers ManualApril 2016

Name

`sparse_keymap_entry_from_scancode` — perform sparse keymap lookup

Synopsis

```
struct key_entry * sparse_keymap_entry_from_scancode (struct
input_dev * dev, unsigned int code);
```

Arguments

dev

Input device using sparse keymap

code

Scan code

Description

This function is used to perform struct key_entry lookup in an input device using sparse keymap.

sparse_keymap_entry_from_keycode

LINUX

Kernel Hackers ManualApril 2016

Name

`sparse_keymap_entry_from_keycode` — perform sparse keymap lookup

Synopsis

```
struct key_entry * sparse_keymap_entry_from_keycode (struct
input_dev * dev, unsigned int keycode);
```

Arguments

dev

Input device using sparse keymap

keycode

Key code

Description

This function is used to perform struct key_entry lookup in an input device using sparse keymap.

sparse_keymap_setup

LINUX

Kernel Hackers ManualApril 2016

Name

`sparse_keymap_setup` — set up sparse keymap for an input device

Synopsis

```
int sparse_keymap_setup (struct input_dev * dev, const struct
key_entry * keymap, int (*setup) (struct input_dev *, struct
key_entry *));
```

Arguments

dev

Input device

keymap

Keymap in form of array of key_entry structures ending with `KE_END` type entry

setup

Function that can be used to adjust keymap entries depending on device's deeds, may be NULL

Description

The function calculates size and allocates copy of the original keymap after which sets up input device event bits appropriately. Before destroying input device allocated keymap should be freed with a call to sparse_keymap_free.

sparse_keymap_free

LINUX

Kernel Hackers ManualApril 2016

Name

sparse_keymap_free — free memory allocated for sparse keymap

Synopsis

```
void sparse_keymap_free (struct input_dev * dev);
```

Arguments

dev

Input device using sparse keymap

Description

This function is used to free memory allocated by sparse keymap in an input device that was set up by `sparse_keymap_setup`.

NOTE

It is safe to cal this function while input device is still registered (however the drivers should care not to try to use freed keymap and thus have to shut off interrups/polling before freeing the keymap).

sparse_keymap_report_entry

LINUX

Kernel Hackers ManualApril 2016

Name

`sparse_keymap_report_entry` — report event corresponding to given key entry

Synopsis

```
void sparse_keymap_report_entry (struct input_dev * dev, const
struct key_entry * ke, unsigned int value, bool autorelease);
```

Arguments

dev

Input device for which event should be reported

ke

key entry describing event

value

Value that should be reported (ignored by KE_SW entries)

autorelease

Signals whether release event should be emitted for KE_KEY entries right after reporting press event, ignored by all other entries

Description

This function is used to report input event described by given struct key_entry.

sparse_keymap_report_event

LINUX

Kernel Hackers ManualApril 2016

Name

sparse_keymap_report_event — report event corresponding to given scancode

Synopsis

```
bool sparse_keymap_report_event (struct input_dev * dev,
unsigned int code, unsigned int value, bool autorelease);
```

Arguments

dev

Input device using sparse keymap

code

Scan code

value

Value that should be reported (ignored by `KE_SW` entries)

autorelease

Signals whether release event should be emitted for `KE_KEY` entries right after reporting press event, ignored by all other entries

Description

This function is used to perform lookup in an input device using sparse keymap and report corresponding event. Returns `true` if lookup was successful and `false` otherwise.

Chapter 9. Serial Peripheral Interface (SPI)

SPI is the "Serial Peripheral Interface", widely used with embedded systems because it is a simple and efficient interface: basically a multiplexed shift register. Its three signal wires hold a clock (SCK, often in the range of 1-20 MHz), a "Master Out, Slave In" (MOSI) data line, and a "Master In, Slave Out" (MISO) data line. SPI is a full duplex protocol; for each bit shifted out the MOSI line (one per clock) another is shifted in on the MISO line. Those bits are assembled into words of various sizes on the way to and from system memory. An additional chipselect line is usually active-low (nCS); four signals are normally used for each peripheral, plus sometimes an interrupt.

The SPI bus facilities listed here provide a generalized interface to declare SPI busses and devices, manage them according to the standard Linux driver model, and perform input/output operations. At this time, only "master" side interfaces are supported, where Linux talks to SPI peripherals and does not implement such a peripheral itself. (Interfaces to support implementing SPI slaves would necessarily look different.)

The programming interface is structured around two kinds of driver, and two kinds of device. A "Controller Driver" abstracts the controller hardware, which may be as simple as a set of GPIO pins or as complex as a pair of FIFOs connected to dual DMA engines on the other side of the SPI shift register (maximizing throughput). Such drivers bridge between whatever bus they sit on (often the platform bus) and SPI, and expose the SPI side of their device as a struct spi_master. SPI devices are children of that master, represented as a struct spi_device and manufactured from struct spi_board_info descriptors which are usually provided by board-specific initialization code. A struct spi_driver is called a "Protocol Driver", and is bound to a spi_device using normal driver model calls.

The I/O model is a set of queued messages. Protocol drivers submit one or more struct spi_message objects, which are processed and completed asynchronously. (There are synchronous wrappers, however.) Messages are built from one or more struct spi_transfer objects, each of which wraps a full duplex SPI transfer. A variety of protocol tweaking options are needed, because different chips adopt very different policies for how they use the bits transferred with SPI.

struct spi_device

LINUX

Name

`struct spi_device` — Master side proxy for an SPI slave device

Synopsis

```
struct spi_device {
  struct device dev;
  struct spi_master * master;
  u32 max_speed_hz;
  u8 chip_select;
  u8 mode;
#define SPI_CPHA 0x01
#define SPI_CPOL 0x02
#define SPI_MODE_0 (0|0)
#define SPI_MODE_1 (0|SPI_CPHA)
#define SPI_MODE_2 (SPI_CPOL|0)
#define SPI_MODE_3 (SPI_CPOL|SPI_CPHA)
#define SPI_CS_HIGH 0x04
#define SPI_LSB_FIRST 0x08
#define SPI_3WIRE 0x10
#define SPI_LOOP 0x20
#define SPI_NO_CS 0x40
#define SPI_READY 0x80
  u8 bits_per_word;
  int irq;
  void * controller_state;
  void * controller_data;
  char modalias[SPI_NAME_SIZE];
};
```

Members

dev

Driver model representation of the device.

master

SPI controller used with the device.

max_speed_hz

Maximum clock rate to be used with this chip (on this board); may be changed by the device's driver. The spi_transfer.speed_hz can override this for each transfer.

chip_select

Chipselect, distinguishing chips handled by *master*.

mode

The spi mode defines how data is clocked out and in. This may be changed by the device's driver. The "active low" default for chipselect mode can be overridden (by specifying SPI_CS_HIGH) as can the "MSB first" default for each word in a transfer (by specifying SPI_LSB_FIRST).

bits_per_word

Data transfers involve one or more words; word sizes like eight or 12 bits are common. In-memory wordsizes are powers of two bytes (e.g. 20 bit samples use 32 bits). This may be changed by the device's driver, or left at the default (0) indicating protocol words are eight bit bytes. The spi_transfer.bits_per_word can override this for each transfer.

irq

Negative, or the number passed to `request_irq` to receive interrupts from this device.

controller_state

Controller's runtime state

controller_data

Board-specific definitions for controller, such as FIFO initialization parameters; from board_info.controller_data

modalias[SPI_NAME_SIZE]

Name of the driver to use with this device, or an alias for that name. This appears in the sysfs "modalias" attribute for driver coldplugging, and in uevents used for hotplugging

Description

A *spi_device* is used to interchange data between an SPI slave (usually a discrete chip) and CPU memory.

In `dev`, the platform_data is used to hold information about this device that's meaningful to the device's protocol driver, but not to its controller. One example might be an identifier for a chip variant with slightly different functionality; another might be information about how this particular board wires the chip's pins.

struct spi_driver

LINUX

Kernel Hackers ManualApril 2016

Name

`struct spi_driver` — Host side "protocol" driver

Synopsis

```
struct spi_driver {
  const struct spi_device_id * id_table;
  int (* probe) (struct spi_device *spi);
  int (* remove) (struct spi_device *spi);
  void (* shutdown) (struct spi_device *spi);
  int (* suspend) (struct spi_device *spi, pm_message_t mesg);
  int (* resume) (struct spi_device *spi);
  struct device_driver driver;
};
```

Members

id_table

List of SPI devices supported by this driver

probe

Binds this driver to the spi device. Drivers can verify that the device is actually present, and may need to configure characteristics (such as bits_per_word) which weren't needed for the initial configuration done during system setup.

remove

Unbinds this driver from the spi device

shutdown

Standard shutdown callback used during system state transitions such as powerdown/halt and kexec

suspend

Standard suspend callback used during system state transitions

resume

Standard resume callback used during system state transitions

driver

SPI device drivers should initialize the name and owner field of this structure.

Description

This represents the kind of device driver that uses SPI messages to interact with the hardware at the other end of a SPI link. It's called a "protocol" driver because it works through messages rather than talking directly to SPI hardware (which is what the underlying SPI controller driver does to pass those messages). These protocols are defined in the specification for the device(s) supported by the driver.

As a rule, those device protocols represent the lowest level interface supported by a driver, and it will support upper level interfaces too. Examples of such upper levels include frameworks like MTD, networking, MMC, RTC, filesystem character device nodes, and hardware monitoring.

spi_unregister_driver

LINUX

Kernel Hackers ManualApril 2016

Name

`spi_unregister_driver` — reverse effect of spi_register_driver

Synopsis

```
void spi_unregister_driver (struct spi_driver * sdrv);
```

Arguments

sdrv

the driver to unregister

Context

can sleep

struct spi_master

LINUX

Kernel Hackers ManualApril 2016

Name

`struct spi_master` — interface to SPI master controller

Synopsis

```
struct spi_master {
  struct device dev;
  struct list_head list;
  s16 bus_num;
  u16 num_chipselect;
  u16 dma_alignment;
  u16 mode_bits;
  u16 flags;
#define SPI_MASTER_HALF_DUPLEX BIT(0)
```

```
#define SPI_MASTER_NO_RX BIT(1)
#define SPI_MASTER_NO_TX BIT(2)
  spinlock_t bus_lock_spinlock;
  struct mutex bus_lock_mutex;
  bool bus_lock_flag;
  int (* setup) (struct spi_device *spi);
  int (* transfer) (struct spi_device *spi,struct spi_message *mesg);
  void (* cleanup) (struct spi_device *spi);
};
```

Members

dev

device interface to this driver

list

link with the global spi_master list

bus_num

board-specific (and often SOC-specific) identifier for a given SPI controller.

num_chipselect

chipselects are used to distinguish individual SPI slaves, and are numbered from zero to num_chipselects. each slave has a chipselect signal, but it's common that not every chipselect is connected to a slave.

dma_alignment

SPI controller constraint on DMA buffers alignment.

mode_bits

flags understood by this controller driver

flags

other constraints relevant to this driver

bus_lock_spinlock

spinlock for SPI bus locking

bus_lock_mutex

mutex for SPI bus locking

bus_lock_flag

indicates that the SPI bus is locked for exclusive use

setup

updates the device mode and clocking records used by a device's SPI controller; protocol code may call this. This must fail if an unrecognized or unsupported mode is requested. It's always safe to call this unless transfers are pending on the device whose settings are being modified.

transfer

adds a message to the controller's transfer queue.

cleanup

frees controller-specific state

Description

Each SPI master controller can communicate with one or more *spi_device* children. These make a small bus, sharing MOSI, MISO and SCK signals but not chip select signals. Each device may be configured to use a different clock rate, since those shared signals are ignored unless the chip is selected.

The driver for an SPI controller manages access to those devices through a queue of spi_message transactions, copying data between CPU memory and an SPI slave device. For each such message it queues, it calls the message's completion function when the transaction completes.

struct spi_transfer

LINUX

Kernel Hackers ManualApril 2016

Name

`struct spi_transfer` — a read/write buffer pair

Synopsis

```
struct spi_transfer {
  const void * tx_buf;
  void * rx_buf;
  unsigned len;
  dma_addr_t tx_dma;
  dma_addr_t rx_dma;
  unsigned cs_change:1;
  u8 bits_per_word;
  u16 delay_usecs;
  u32 speed_hz;
  struct list_head transfer_list;
};
```

Members

tx_buf

data to be written (dma-safe memory), or NULL

rx_buf

data to be read (dma-safe memory), or NULL

len

size of rx and tx buffers (in bytes)

tx_dma

DMA address of tx_buf, if *spi_message*.is_dma_mapped

rx_dma

DMA address of rx_buf, if *spi_message*.is_dma_mapped

cs_change

affects chipselect after this transfer completes

bits_per_word

select a bits_per_word other than the device default for this transfer. If 0 the default (from *spi_device*) is used.

delay_usecs

microseconds to delay after this transfer before (optionally) changing the chipselect status, then starting the next transfer or completing this *spi_message*.

speed_hz

Select a speed other than the device default for this transfer. If 0 the default (from *spi_device*) is used.

transfer_list

transfers are sequenced through *spi_message*.transfers

Description

SPI transfers always write the same number of bytes as they read. Protocol drivers should always provide *rx_buf* and/or *tx_buf*. In some cases, they may also want to provide DMA addresses for the data being transferred; that may reduce overhead, when the underlying driver uses dma.

If the transmit buffer is null, zeroes will be shifted out while filling *rx_buf*. If the receive buffer is null, the data shifted in will be discarded. Only “len” bytes shift out (or in). It’s an error to try to shift out a partial word. (For example, by shifting out three bytes with word size of sixteen or twenty bits; the former uses two bytes per word, the latter uses four bytes.)

In-memory data values are always in native CPU byte order, translated from the wire byte order (big-endian except with SPI_LSB_FIRST). So for example when bits_per_word is sixteen, buffers are 2N bytes long (*len* = 2N) and hold N sixteen bit words in CPU byte order.

When the word size of the SPI transfer is not a power-of-two multiple of eight bits, those in-memory words include extra bits. In-memory words are always seen by protocol drivers as right-justified, so the undefined (rx) or unused (tx) bits are always the most significant bits.

All SPI transfers start with the relevant chipselect active. Normally it stays selected until after the last transfer in a message. Drivers can affect the chipselect signal using cs_change.

(i) If the transfer isn’t the last one in the message, this flag is used to make the chipselect briefly go inactive in the middle of the message. Toggling chipselect in this way may be needed to terminate a chip command, letting a single spi_message perform all of group of chip transactions together.

(ii) When the transfer is the last one in the message, the chip may stay selected until the next transfer. On multi-device SPI busses with nothing blocking messages going to other devices, this is just a performance hint; starting a message to another device deselects this one. But in other cases, this can be used to ensure correctness. Some devices need protocol transactions to be built from a series of spi_message submissions, where the content of one message is determined by the results of previous messages and where the whole transaction ends when the chipselect goes intactive.

The code that submits an spi_message (and its spi_transfers) to the lower layers is responsible for managing its memory. Zero-initialize every field you don't set up explicitly, to insulate against future API updates. After you submit a message and its transfers, ignore them until its completion callback.

struct spi_message

LINUX

Kernel Hackers ManualApril 2016

Name

`struct spi_message` — one multi-segment SPI transaction

Synopsis

```
struct spi_message {
  struct list_head transfers;
  struct spi_device * spi;
  unsigned is_dma_mapped:1;
  void (* complete) (void *context);
  void * context;
  unsigned actual_length;
  int status;
  struct list_head queue;
  void * state;
};
```

Members

transfers

list of transfer segments in this transaction

spi

SPI device to which the transaction is queued

is_dma_mapped

if true, the caller provided both dma and cpu virtual addresses for each transfer buffer

complete

called to report transaction completions

context

the argument to `complete` when it's called

actual_length

the total number of bytes that were transferred in all successful segments

status

zero for success, else negative errno

queue

for use by whichever driver currently owns the message

state

for use by whichever driver currently owns the message

Description

A *`spi_message`* is used to execute an atomic sequence of data transfers, each represented by a struct spi_transfer. The sequence is "atomic" in the sense that no other spi_message may use that SPI bus until that sequence completes. On some systems, many such sequences can execute as as single programmed DMA transfer. On all systems, these messages are queued, and might complete after transactions to other devices. Messages sent to a given spi_device are alway executed in FIFO order.

The code that submits an spi_message (and its spi_transfers) to the lower layers is responsible for managing its memory. Zero-initialize every field you don't set up explicitly, to insulate against future API updates. After you submit a message and its transfers, ignore them until its completion callback.

spi_write

LINUX

Kernel Hackers ManualApril 2016

Name

`spi_write` — SPI synchronous write

Synopsis

```
int spi_write (struct spi_device * spi, const void * buf,
size_t len);
```

Arguments

spi

device to which data will be written

buf

data buffer

len

data buffer size

Context

can sleep

Description

This writes the buffer and returns zero or a negative error code. Callable only from contexts that can sleep.

spi_read

LINUX

Kernel Hackers ManualApril 2016

Name

`spi_read` — SPI synchronous read

Synopsis

```
int spi_read (struct spi_device * spi, void * buf, size_t
len);
```

Arguments

spi

device from which data will be read

buf

data buffer

len

data buffer size

Context

can sleep

Description

This reads the buffer and returns zero or a negative error code. Callable only from contexts that can sleep.

spi_w8r8

LINUX

Kernel Hackers ManualApril 2016

Name

spi_w8r8 — SPI synchronous 8 bit write followed by 8 bit read

Synopsis

```
ssize_t spi_w8r8 (struct spi_device * spi, u8 cmd);
```

Arguments

spi

device with which data will be exchanged

cmd

command to be written before data is read back

Context

can sleep

Description

This returns the (unsigned) eight bit number returned by the device, or else a negative error code. Callable only from contexts that can sleep.

spi_w8r16

LINUX

Kernel Hackers ManualApril 2016

Name

`spi_w8r16` — SPI synchronous 8 bit write followed by 16 bit read

Synopsis

```
ssize_t spi_w8r16 (struct spi_device * spi, u8 cmd);
```

Arguments

spi

device with which data will be exchanged

cmd

command to be written before data is read back

Context

can sleep

Description

This returns the (unsigned) sixteen bit number returned by the device, or else a negative error code. Callable only from contexts that can sleep.

The number is returned in wire-order, which is at least sometimes big-endian.

struct spi_board_info

LINUX

Kernel Hackers ManualApril 2016

Name

`struct spi_board_info` — board-specific template for a SPI device

Synopsis

```
struct spi_board_info {
  char modalias[SPI_NAME_SIZE];
  const void * platform_data;
  void * controller_data;
  int irq;
  u32 max_speed_hz;
  u16 bus_num;
  u16 chip_select;
  u8 mode;
};
```

Members

modalias[SPI_NAME_SIZE]

Initializes spi_device.modalias; identifies the driver.

platform_data

Initializes spi_device.platform_data; the particular data stored there is driver-specific.

controller_data

Initializes spi_device.controller_data; some controllers need hints about hardware setup, e.g. for DMA.

irq

Initializes spi_device.irq; depends on how the board is wired.

max_speed_hz

Initializes spi_device.max_speed_hz; based on limits from the chip datasheet and board-specific signal quality issues.

bus_num

Identifies which spi_master parents the spi_device; unused by `spi_new_device`, and otherwise depends on board wiring.

chip_select

Initializes spi_device.chip_select; depends on how the board is wired.

mode

Initializes spi_device.mode; based on the chip datasheet, board wiring (some devices support both 3WIRE and standard modes), and possibly presence of an inverter in the chipselect path.

Description

When adding new SPI devices to the device tree, these structures serve as a partial device template. They hold information which can’t always be determined by drivers. Information that `probe` can establish (such as the default transfer wordsize) is not included here.

These structures are used in two places. Their primary role is to be stored in tables of board-specific device descriptors, which are declared early in board initialization

and then used (much later) to populate a controller's device tree after the that controller's driver initializes. A secondary (and atypical) role is as a parameter to `spi_new_device` call, which happens after those controller drivers are active in some dynamic board configuration models.

spi_register_board_info

LINUX

Kernel Hackers ManualApril 2016

Name

`spi_register_board_info` — register SPI devices for a given board

Synopsis

```
int spi_register_board_info (struct spi_board_info const *
info, unsigned n);
```

Arguments

info

array of chip descriptors

n

how many descriptors are provided

Context

can sleep

Description

Board-specific early init code calls this (probably during arch_initcall) with segments of the SPI device table. Any device nodes are created later, after the relevant parent SPI controller (bus_num) is defined. We keep this table of devices forever, so that reloading a controller driver will not make Linux forget about these hard-wired devices.

Other code can also call this, e.g. a particular add-on board might provide SPI devices through its expansion connector, so code initializing that board would naturally declare its SPI devices.

The board info passed can safely be __initdata ... but be careful of any embedded pointers (platform_data, etc), they're copied as-is.

spi_register_driver

LINUX

Kernel Hackers ManualApril 2016

Name

`spi_register_driver` — register a SPI driver

Synopsis

```
int spi_register_driver (struct spi_driver * sdrv);
```

Arguments

`sdrv`

the driver to register

Context

can sleep

spi_alloc_device

LINUX

Kernel Hackers ManualApril 2016

Name

`spi_alloc_device` — Allocate a new SPI device

Synopsis

```
struct spi_device * spi_alloc_device (struct spi_master *
master);
```

Arguments

master

Controller to which device is connected

Context

can sleep

Description

Allows a driver to allocate and initialize a spi_device without registering it immediately. This allows a driver to directly fill the spi_device with device parameters before calling `spi_add_device` on it.

Caller is responsible to call `spi_add_device` on the returned spi_device structure to add it to the SPI master. If the caller needs to discard the spi_device without adding it, then it should call `spi_dev_put` on it.

Returns a pointer to the new device, or NULL.

spi_add_device

LINUX

Kernel Hackers ManualApril 2016

Name

`spi_add_device` — Add spi_device allocated with spi_alloc_device

Synopsis

```
int spi_add_device (struct spi_device * spi);
```

Arguments

spi

spi_device to register

Description

Companion function to spi_alloc_device. Devices allocated with spi_alloc_device can be added onto the spi bus with this function.

Returns 0 on success; negative errno on failure

spi_new_device

LINUX

Kernel Hackers ManualApril 2016

Name

`spi_new_device` — instantiate one new SPI device

Synopsis

```
struct spi_device * spi_new_device (struct spi_master *
master, struct spi_board_info * chip);
```

Arguments

master

Controller to which device is connected

chip

Describes the SPI device

Context

can sleep

Description

On typical mainboards, this is purely internal; and it's not needed after board init creates the hard-wired devices. Some development platforms may not be able to use spi_register_board_info though, and this is exported so that for example a USB or parport based adapter driver could add devices (which it would learn about out-of-band).

Returns the new device, or NULL.

spi_alloc_master

LINUX

Kernel Hackers ManualApril 2016

Name

`spi_alloc_master` — allocate SPI master controller

Synopsis

```
struct spi_master * spi_alloc_master (struct device * dev,
unsigned size);
```

Arguments

dev

the controller, possibly using the platform_bus

size

how much zeroed driver-private data to allocate; the pointer to this memory is in the driver_data field of the returned device, accessible with `spi_master_get_devdata`.

Context

can sleep

Description

This call is used only by SPI master controller drivers, which are the only ones directly touching chip registers. It's how they allocate an spi_master structure, prior to calling `spi_register_master`.

This must be called from context that can sleep. It returns the SPI master structure on success, else NULL.

The caller is responsible for assigning the bus number and initializing the master's methods before calling `spi_register_master`; and (after errors adding the device) calling `spi_master_put` to prevent a memory leak.

spi_register_master

LINUX

Kernel Hackers ManualApril 2016

Name

`spi_register_master` — register SPI master controller

Synopsis

```
int spi_register_master (struct spi_master * master);
```

Arguments

master

initialized master, originally from spi_alloc_master

Context

can sleep

Description

SPI master controllers connect to their drivers using some non-SPI bus, such as the platform bus. The final stage of probe in that code includes calling spi_register_master to hook up to this SPI bus glue.

SPI controllers use board specific (often SOC specific) bus numbers, and board-specific addressing for SPI devices combines those numbers with chip select numbers. Since SPI does not directly support dynamic device identification, boards need configuration tables telling which chip is at which address.

This must be called from context that can sleep. It returns zero on success, else a negative error code (dropping the master's refcount). After a successful return, the caller is responsible for calling spi_unregister_master.

spi_unregister_master

LINUX

Kernel Hackers ManualApril 2016

Name

spi_unregister_master — unregister SPI master controller

Synopsis

```
void spi_unregister_master (struct spi_master * master);
```

Arguments

master

the master being unregistered

Context

can sleep

Description

This call is used only by SPI master controller drivers, which are the only ones directly touching chip registers.

This must be called from context that can sleep.

spi_busnum_to_master

LINUX

Kernel Hackers ManualApril 2016

Name

`spi_busnum_to_master` — look up master associated with bus_num

Synopsis

```
struct spi_master * spi_busnum_to_master (u16 bus_num);
```

Arguments

bus_num

the master's bus number

Context

can sleep

Description

This call may be used with devices that are registered after arch init time. It returns a refcounted pointer to the relevant spi_master (which the caller must release), or NULL if there is no such master registered.

spi_setup

LINUX

Kernel Hackers ManualApril 2016

Name

`spi_setup` — setup SPI mode and clock rate

Synopsis

```
int spi_setup (struct spi_device * spi);
```

Arguments

spi

the device whose settings are being modified

Context

can sleep, and no requests are queued to the device

Description

SPI protocol drivers may need to update the transfer mode if the device doesn't work with its default. They may likewise need to update clock rates or word sizes from initial values. This function changes those settings, and must be called from a context that can sleep. Except for SPI_CS_HIGH, which takes effect immediately, the changes take effect the next time the device is selected and data is transferred to or from it. When this function returns, the spi device is deselected.

Note that this call will fail if the protocol driver specifies an option that the underlying controller or its driver does not support. For example, not all hardware supports wire transfers using nine bit words, LSB-first wire encoding, or active-high chipselects.

spi_async

LINUX

Name

`spi_async` — asynchronous SPI transfer

Synopsis

```
int spi_async (struct spi_device * spi, struct spi_message *
message);
```

Arguments

`spi`

device with which data will be exchanged

`message`

describes the data transfers, including completion callback

Context

any (irqs may be blocked, etc)

Description

This call may be used in_irq and other contexts which can't sleep, as well as from task contexts which can sleep.

The completion callback is invoked in a context which can't sleep. Before that invocation, the value of message->status is undefined. When the callback is issued, message->status holds either zero (to indicate complete success) or a negative error code. After that callback returns, the driver which issued the transfer request may deallocate the associated memory; it's no longer in use by any SPI core or controller driver code.

Note that although all messages to a spi_device are handled in FIFO order, messages may go to different devices in other orders. Some device might be higher priority, or have various “hard” access time requirements, for example.

On detection of any fault during the transfer, processing of the entire message is aborted, and the device is deselected. Until returning from the associated message completion callback, no other spi_message queued to that device will be processed. (This rule applies equally to all the synchronous transfer calls, which are wrappers around this core asynchronous primitive.)

spi_async_locked

LINUX

Kernel Hackers ManualApril 2016

Name

`spi_async_locked` — version of spi_async with exclusive bus usage

Synopsis

```
int spi_async_locked (struct spi_device * spi, struct
spi_message * message);
```

Arguments

spi

device with which data will be exchanged

message

describes the data transfers, including completion callback

Context

any (irqs may be blocked, etc)

Description

This call may be used in_irq and other contexts which can't sleep, as well as from task contexts which can sleep.

The completion callback is invoked in a context which can't sleep. Before that invocation, the value of message->status is undefined. When the callback is issued, message->status holds either zero (to indicate complete success) or a negative error code. After that callback returns, the driver which issued the transfer request may deallocate the associated memory; it's no longer in use by any SPI core or controller driver code.

Note that although all messages to a spi_device are handled in FIFO order, messages may go to different devices in other orders. Some device might be higher priority, or have various "hard" access time requirements, for example.

On detection of any fault during the transfer, processing of the entire message is aborted, and the device is deselected. Until returning from the associated message completion callback, no other spi_message queued to that device will be processed. (This rule applies equally to all the synchronous transfer calls, which are wrappers around this core asynchronous primitive.)

spi_sync

LINUX

Kernel Hackers ManualApril 2016

Name

`spi_sync` — blocking/synchronous SPI data transfers

Synopsis

```
int spi_sync (struct spi_device * spi, struct spi_message *
message);
```

Arguments

spi

device with which data will be exchanged

message

describes the data transfers

Context

can sleep

Description

This call may only be used from a context that may sleep. The sleep is non-interruptible, and has no timeout. Low-overhead controller drivers may DMA directly into and out of the message buffers.

Note that the SPI device's chip select is active during the message, and then is normally disabled between messages. Drivers for some frequently-used devices may want to minimize costs of selecting a chip, by leaving it selected in anticipation that the next message will go to the same chip. (That may increase power usage.)

Also, the caller is guaranteeing that the memory associated with the message will not be freed before this call returns.

It returns zero on success, else a negative error code.

spi_sync_locked

LINUX

Kernel Hackers ManualApril 2016

Name

`spi_sync_locked` — version of spi_sync with exclusive bus usage

Synopsis

```
int spi_sync_locked (struct spi_device * spi, struct
spi_message * message);
```

Arguments

spi

device with which data will be exchanged

message

describes the data transfers

Context

can sleep

Description

This call may only be used from a context that may sleep. The sleep is non-interruptible, and has no timeout. Low-overhead controller drivers may DMA directly into and out of the message buffers.

This call should be used by drivers that require exclusive access to the SPI bus. It has to be preceded by a spi_bus_lock call. The SPI bus must be released by a spi_bus_unlock call when the exclusive access is over.

It returns zero on success, else a negative error code.

spi_bus_lock

LINUX

Kernel Hackers ManualApril 2016

Name

`spi_bus_lock` — obtain a lock for exclusive SPI bus usage

Synopsis

```
int spi_bus_lock (struct spi_master * master);
```

Arguments

master

SPI bus master that should be locked for exclusive bus access

Context

can sleep

Description

This call may only be used from a context that may sleep. The sleep is non-interruptible, and has no timeout.

This call should be used by drivers that require exclusive access to the SPI bus. The SPI bus must be released by a spi_bus_unlock call when the exclusive access is

over. Data transfer must be done by spi_sync_locked and spi_async_locked calls when the SPI bus lock is held.

It returns zero on success, else a negative error code.

spi_bus_unlock

LINUX

Kernel Hackers ManualApril 2016

Name

`spi_bus_unlock` — release the lock for exclusive SPI bus usage

Synopsis

```
int spi_bus_unlock (struct spi_master * master);
```

Arguments

master

SPI bus master that was locked for exclusive bus access

Context

can sleep

Description

This call may only be used from a context that may sleep. The sleep is non-interruptible, and has no timeout.

This call releases an SPI bus lock previously obtained by an spi_bus_lock call.

It returns zero on success, else a negative error code.

spi_write_then_read

LINUX

Kernel Hackers ManualApril 2016

Name

`spi_write_then_read` — SPI synchronous write followed by read

Synopsis

```
int spi_write_then_read (struct spi_device * spi, const void *
txbuf, unsigned n_tx, void * rxbuf, unsigned n_rx);
```

Arguments

spi

device with which data will be exchanged

txbuf

data to be written (need not be dma-safe)

n_tx

size of txbuf, in bytes

rxbuf

buffer into which data will be read (need not be dma-safe)

`n_rx`

size of rxbuf, in bytes

Context

can sleep

Description

This performs a half duplex MicroWire style transaction with the device, sending txbuf and then reading rxbuf. The return value is zero for success, else a negative errno status code. This call may only be used from a context that may sleep.

Parameters to this routine are always copied using a small buffer; portable code should never use this for more than 32 bytes. Performance-sensitive or bulk transfer code should instead use spi_{async,sync}() calls with dma-safe buffers.

Chapter 10. I²C and SMBus Subsystem

I²C (or without fancy typography, "I2C") is an acronym for the "Inter-IC" bus, a simple bus protocol which is widely used where low data rate communications suffice. Since it's also a licensed trademark, some vendors use another name (such as "Two-Wire Interface", TWI) for the same bus. I2C only needs two signals (SCL for clock, SDA for data), conserving board real estate and minimizing signal quality issues. Most I2C devices use seven bit addresses, and bus speeds of up to 400 kHz; there's a high speed extension (3.4 MHz) that's not yet found wide use. I2C is a multi-master bus; open drain signaling is used to arbitrate between masters, as well as to handshake and to synchronize clocks from slower clients.

The Linux I2C programming interfaces support only the master side of bus interactions, not the slave side. The programming interface is structured around two kinds of driver, and two kinds of device. An I2C "Adapter Driver" abstracts the controller hardware; it binds to a physical device (perhaps a PCI device or platform_device) and exposes a struct i2c_adapter representing each I2C bus segment it manages. On each I2C bus segment will be I2C devices represented by a struct i2c_client. Those devices will be bound to a struct i2c_driver, which should follow the standard Linux driver model. (At this writing, a legacy model is more widely used.) There are functions to perform various I2C protocol operations; at this writing all such functions are usable only from task context.

The System Management Bus (SMBus) is a sibling protocol. Most SMBus systems are also I2C conformant. The electrical constraints are tighter for SMBus, and it standardizes particular protocol messages and idioms. Controllers that support I2C can also support most SMBus operations, but SMBus controllers don't support all the protocol options that an I2C controller will. There are functions to perform various SMBus protocol operations, either using I2C primitives or by issuing SMBus commands to i2c_adapter devices which don't support those I2C operations.

struct i2c_driver

LINUX

Kernel Hackers ManualApril 2016

Name

`struct i2c_driver` — represent an I2C device driver

Synopsis

```
struct i2c_driver {
  unsigned int class;
  int (* attach_adapter) (struct i2c_adapter *);
  int (* detach_adapter) (struct i2c_adapter *);
  int (* probe) (struct i2c_client *, const struct i2c_device_id *);
  int (* remove) (struct i2c_client *);
  void (* shutdown) (struct i2c_client *);
  int (* suspend) (struct i2c_client *, pm_message_t mesg);
  int (* resume) (struct i2c_client *);
  void (* alert) (struct i2c_client *, unsigned int data);
  int (* command) (struct i2c_client *client, unsigned int cmd, void *arg)
  struct device_driver driver;
  const struct i2c_device_id * id_table;
  int (* detect) (struct i2c_client *, struct i2c_board_info *);
  const unsigned short * address_list;
  struct list_head clients;
};
```

Members

class

What kind of i2c device we instantiate (for detect)

attach_adapter

Callback for bus addition (deprecated)

detach_adapter

Callback for bus removal (deprecated)

probe

Callback for device binding

remove

Callback for device unbinding

shutdown

Callback for device shutdown

suspend

Callback for device suspend

resume

Callback for device resume

alert

Alert callback, for example for the SMBus alert protocol

command

Callback for bus-wide signaling (optional)

driver

Device driver model driver

id_table

List of I2C devices supported by this driver

detect

Callback for device detection

address_list

The I2C addresses to probe (for detect)

clients

List of detected clients we created (for i2c-core use only)

Description

The driver.owner field should be set to the module owner of this driver. The driver.name field should be set to the name of this driver.

For automatic device detection, both `detect` and `address_data` must be defined. `class` should also be set, otherwise only devices forced with module parameters will be created. The detect function must fill at least the name field of the i2c_board_info structure it is handed upon successful detection, and possibly also the flags field.

If `detect` is missing, the driver will still work fine for enumerated devices. Detected devices simply won't be supported. This is expected for the many I2C/SMBus devices which can't be detected reliably, and the ones which can always be enumerated in practice.

The i2c_client structure which is handed to the `detect` callback is not a real i2c_client. It is initialized just enough so that you can call i2c_smbus_read_byte_data and friends on it. Don't do anything else with it. In particular, calling dev_dbg and friends on it is not allowed.

struct i2c_client

LINUX

Kernel Hackers ManualApril 2016

Name

`struct i2c_client` — represent an I2C slave device

Synopsis

```
struct i2c_client {
  unsigned short flags;
  unsigned short addr;
  char name[I2C_NAME_SIZE];
  struct i2c_adapter * adapter;
  struct i2c_driver * driver;
  struct device dev;
  int irq;
  struct list_head detected;
};
```

Members

flags

I2C_CLIENT_TEN indicates the device uses a ten bit chip address; I2C_CLIENT_PEC indicates it uses SMBus Packet Error Checking

addr

Address used on the I2C bus connected to the parent adapter.

name[I2C_NAME_SIZE]

Indicates the type of the device, usually a chip name that's generic enough to hide second-sourcing and compatible revisions.

adapter

manages the bus segment hosting this I2C device

driver

device's driver, hence pointer to access routines

dev

Driver model device node for the slave.

irq

indicates the IRQ generated by this device (if any)

detected

member of an i2c_driver.clients list or i2c-core's userspace_devices list

Description

An i2c_client identifies a single device (i.e. chip) connected to an i2c bus. The behaviour exposed to Linux is defined by the driver managing the device.

struct i2c_board_info

LINUX

Kernel Hackers ManualApril 2016

Name

`struct i2c_board_info` — template for device creation

Synopsis

```
struct i2c_board_info {
  char type[I2C_NAME_SIZE];
  unsigned short flags;
  unsigned short addr;
  void * platform_data;
  struct dev_archdata * archdata;
  struct device_node * of_node;
  int irq;
};
```

Members

type[I2C_NAME_SIZE]

chip type, to initialize i2c_client.name

flags

to initialize i2c_client.flags

addr

stored in i2c_client.addr

platform_data

stored in i2c_client.dev.platform_data

archdata

copied into i2c_client.dev.archdata

of_node

pointer to OpenFirmware device node

irq

stored in i2c_client.irq

Description

I2C doesn't actually support hardware probing, although controllers and devices may be able to use I2C_SMBUS_QUICK to tell whether or not there's a device at a

given address. Drivers commonly need more information than that, such as chip type, configuration, associated IRQ, and so on.

i2c_board_info is used to build tables of information listing I2C devices that are present. This information is used to grow the driver model tree. For mainboards this is done statically using `i2c_register_board_info`; bus numbers identify adapters that aren't yet available. For add-on boards, `i2c_new_device` does this dynamically with the adapter already known.

I2C_BOARD_INFO

LINUX

Kernel Hackers ManualApril 2016

Name

`I2C_BOARD_INFO` — macro used to list an i2c device and its address

Synopsis

```
I2C_BOARD_INFO ( dev_type,  dev_addr);
```

Arguments

`dev_type`

identifies the device type

`dev_addr`

the device's address on the bus.

Description

This macro initializes essential fields of a struct i2c_board_info, declaring what has been provided on a particular board. Optional fields (such as associated irq, or device-specific platform_data) are provided using conventional syntax.

struct i2c_msg

LINUX

Kernel Hackers ManualApril 2016

Name

`struct i2c_msg` — an I2C transaction segment beginning with START

Synopsis

```
struct i2c_msg {
  __u16 addr;
  __u16 flags;
#define I2C_M_TEN  0x0010
#define I2C_M_RD  0x0001
#define I2C_M_NOSTART  0x4000
#define I2C_M_REV_DIR_ADDR 0x2000
#define I2C_M_IGNORE_NAK 0x1000
#define I2C_M_NO_RD_ACK  0x0800
#define I2C_M_RECV_LEN  0x0400
  __u16 len;
  __u8 * buf;
};
```

Members

addr

Slave address, either seven or ten bits. When this is a ten bit address, I2C_M_TEN must be set in *`flags`* and the adapter must support

I2C_FUNC_10BIT_ADDR.

flags

I2C_M_RD is handled by all adapters. No other flags may be provided unless the adapter exported the relevant I2C_FUNC_* flags through `i2c_check_functionality`.

len

Number of data bytes in *`buf`* being read from or written to the I2C slave address. For read transactions where I2C_M_RECV_LEN is set, the caller guarantees that this buffer can hold up to 32 bytes in addition to the initial length byte sent by the slave (plus, if used, the SMBus PEC); and this value will be incremented by the number of block data bytes received.

buf

The buffer into which data is read, or from which it's written.

Description

An i2c_msg is the low level representation of one segment of an I2C transaction. It is visible to drivers in the *`i2c_transfer`*() procedure, to userspace from i2c-dev, and to I2C adapter drivers through the *`i2c_adapter.master_xfer`*() method.

Except when I2C "protocol mangling" is used, all I2C adapters implement the standard rules for I2C transactions. Each transaction begins with a START. That is followed by the slave address, and a bit encoding read versus write. Then follow all the data bytes, possibly including a byte with SMBus PEC. The transfer terminates with a NAK, or when all those bytes have been transferred and ACKed. If this is the last message in a group, it is followed by a STOP. Otherwise it is followed by the next *`i2c_msg`* transaction segment, beginning with a (repeated) START.

Alternatively, when the adapter supports I2C_FUNC_PROTOCOL_MANGLING then passing certain *`flags`* may have changed those standard protocol behaviors. Those flags are only for use with broken/nonconforming slaves, and with adapters which are known to support the specific mangling options they need (one or more of IGNORE_NAK, NO_RD_ACK, NOSTART, and REV_DIR_ADDR).

i2c_register_board_info

LINUX

Kernel Hackers ManualApril 2016

Name

`i2c_register_board_info` — statically declare I2C devices

Synopsis

```
int i2c_register_board_info (int busnum, struct i2c_board_info
const * info, unsigned len);
```

Arguments

busnum

identifies the bus to which these devices belong

info

vector of i2c device descriptors

len

how many descriptors in the vector; may be zero to reserve the specified bus number.

Description

Systems using the Linux I2C driver stack can declare tables of board info while they initialize. This should be done in board-specific init code near `arch_initcall` time, or equivalent, before any I2C adapter driver is registered. For example, mainboard init code could define several devices, as could the init code for each daughtercard in a board stack.

The I2C devices will be created later, after the adapter for the relevant bus has been registered. After that moment, standard driver model tools are used to bind “new style” I2C drivers to the devices. The bus number for any device declared using this routine is not available for dynamic allocation.

The board info passed can safely be __initdata, but be careful of embedded pointers (for platform_data, functions, etc) since that won’t be copied.

i2c_verify_client

LINUX

Kernel Hackers ManualApril 2016

Name

`i2c_verify_client` — return parameter as i2c_client, or NULL

Synopsis

```
struct i2c_client * i2c_verify_client (struct device * dev);
```

Arguments

dev

device, probably from some driver model iterator

Description

When traversing the driver model tree, perhaps using driver model iterators like *device_for_each_child*(), you can’t assume very much about the nodes you find. Use this function to avoid oopses caused by wrongly treating some non-I2C device as an i2c_client.

i2c_lock_adapter

LINUX

Kernel Hackers ManualApril 2016

Name

`i2c_lock_adapter` — Get exclusive access to an I2C bus segment

Synopsis

```
void i2c_lock_adapter (struct i2c_adapter * adapter);
```

Arguments

adapter

Target I2C bus segment

i2c_unlock_adapter

LINUX

Kernel Hackers ManualApril 2016

Name

`i2c_unlock_adapter` — Release exclusive access to an I2C bus segment

Synopsis

```
void i2c_unlock_adapter (struct i2c_adapter * adapter);
```

Arguments

adapter

Target I2C bus segment

i2c_new_device

LINUX

Kernel Hackers ManualApril 2016

Name

i2c_new_device — instantiate an i2c device

Synopsis

```
struct i2c_client * i2c_new_device (struct i2c_adapter * adap,
struct i2c_board_info const * info);
```

Arguments

adap

the adapter managing the device

info

describes one I2C device; bus_num is ignored

Context

can sleep

Description

Create an i2c device. Binding is handled through driver model `probe`/`remove` methods. A driver may be bound to this device when we return from this function, or any later moment (e.g. maybe hotplugging will load the driver module). This call is not appropriate for use by mainboard initialization logic, which usually runs during an `arch_initcall` long before any i2c_adapter could exist.

This returns the new i2c client, which may be saved for later use with `i2c_unregister_device`; or NULL to indicate an error.

i2c_unregister_device

LINUX

Kernel Hackers ManualApril 2016

Name

`i2c_unregister_device` — reverse effect of `i2c_new_device`

Synopsis

```
void i2c_unregister_device (struct i2c_client * client);
```

Arguments

client

value returned from i2c_new_device

Context

can sleep

i2c_new_dummy

LINUX

Kernel Hackers ManualApril 2016

Name

i2c_new_dummy — return a new i2c device bound to a dummy driver

Synopsis

```
struct i2c_client * i2c_new_dummy (struct i2c_adapter *
adapter, u16 address);
```

Arguments

adapter

the adapter managing the device

address

seven bit address to be used

Context

can sleep

Description

This returns an I2C client bound to the “dummy” driver, intended for use with devices that consume multiple addresses. Examples of such chips include various EEPROMS (like 24c04 and 24c08 models).

These dummy devices have two main uses. First, most I2C and SMBus calls except `i2c_transfer` need a client handle; the dummy will be that handle. And second, this prevents the specified address from being bound to a different driver.

This returns the new i2c client, which should be saved for later use with `i2c_unregister_device`; or NULL to indicate an error.

i2c_add_adapter

LINUX

Kernel Hackers ManualApril 2016

Name

`i2c_add_adapter` — declare i2c adapter, use dynamic bus number

Synopsis

```
int i2c_add_adapter (struct i2c_adapter * adapter);
```

Arguments

adapter

the adapter to add

Context

can sleep

Description

This routine is used to declare an I2C adapter when its bus number doesn't matter. Examples: for I2C adapters dynamically added by USB links or PCI plugin cards.

When this returns zero, a new bus number was allocated and stored in adap->nr, and the specified adapter became available for clients. Otherwise, a negative errno value is returned.

i2c_add_numbered_adapter

LINUX

Kernel Hackers ManualApril 2016

Name

`i2c_add_numbered_adapter` — declare i2c adapter, use static bus number

Synopsis

```
int i2c_add_numbered_adapter (struct i2c_adapter * adap);
```

Arguments

adap

the adapter to register (with adap->nr initialized)

Context

can sleep

Description

This routine is used to declare an I2C adapter when its bus number matters. For example, use it for I2C adapters from system-on-chip CPUs, or otherwise built in to the system's mainboard, and where i2c_board_info is used to properly configure I2C devices.

If no devices have pre-been declared for this bus, then be sure to register the adapter before any dynamically allocated ones. Otherwise the required bus ID may not be available.

When this returns zero, the specified adapter became available for clients using the bus number provided in adap->nr. Also, the table of I2C devices pre-declared using `i2c_register_board_info` is scanned, and the appropriate driver model device nodes are created. Otherwise, a negative errno value is returned.

i2c_del_adapter

LINUX

Kernel Hackers ManualApril 2016

Name

`i2c_del_adapter` — unregister I2C adapter

Synopsis

```
int i2c_del_adapter (struct i2c_adapter * adap);
```

Arguments

adap

the adapter being unregistered

Context

can sleep

Description

This unregisters an I2C adapter which was previously registered by *i2c_add_adapter* or *i2c_add_numbered_adapter*.

i2c_del_driver

LINUX

Kernel Hackers ManualApril 2016

Name

i2c_del_driver — unregister I2C driver

Synopsis

```
void i2c_del_driver (struct i2c_driver * driver);
```

Arguments

driver

the driver being unregistered

Context

can sleep

i2c_use_client

LINUX

Kernel Hackers ManualApril 2016

Name

`i2c_use_client` — increments the reference count of the i2c client structure

Synopsis

```
struct i2c_client * i2c_use_client (struct i2c_client *
client);
```

Arguments

client

the client being referenced

Description

Each live reference to a client should be refcounted. The driver model does that automatically as part of driver binding, so that most drivers don't

need to do this explicitly

they hold a reference until they're unbound from the device.

A pointer to the client with the incremented reference counter is returned.

i2c_release_client

LINUX

Kernel Hackers ManualApril 2016

Name

`i2c_release_client` — release a use of the i2c client structure

Synopsis

```
void i2c_release_client (struct i2c_client * client);
```

Arguments

client

the client being no longer referenced

Description

Must be called when a user of a client is finished with it.

i2c_transfer

LINUX

Kernel Hackers ManualApril 2016

Name

`i2c_transfer` — execute a single or combined I2C message

Synopsis

```
int i2c_transfer (struct i2c_adapter * adap, struct i2c_msg *
msgs, int num);
```

Arguments

adap

Handle to I2C bus

msgs

One or more messages to execute before STOP is issued to terminate the operation; each message begins with a START.

num

Number of messages to be executed.

Description

Returns negative errno, else the number of messages executed.

Note that there is no requirement that each message be sent to the same slave address, although that is the most common model.

i2c_master_send

LINUX

Kernel Hackers ManualApril 2016

Name

`i2c_master_send` — issue a single I2C message in master transmit mode

Synopsis

```
int i2c_master_send (const struct i2c_client * client, const
char * buf, int count);
```

Arguments

client

Handle to slave device

buf

Data that will be written to the slave

count

How many bytes to write, must be less than 64k since msg.len is u16

Description

Returns negative errno, or else the number of bytes written.

i2c_master_recv

LINUX

Kernel Hackers ManualApril 2016

Name

`i2c_master_recv` — issue a single I2C message in master receive mode

Synopsis

```
int i2c_master_recv (const struct i2c_client * client, char *
buf, int count);
```

Arguments

client

Handle to slave device

buf

Where to store data read from slave

count

How many bytes to read, must be less than 64k since msg.len is u16

Description

Returns negative errno, or else the number of bytes read.

i2c_smbus_read_byte

LINUX

Kernel Hackers ManualApril 2016

Name

`i2c_smbus_read_byte` — SMBus “receive byte” protocol

Synopsis

```
s32 i2c_smbus_read_byte (const struct i2c_client * client);
```

Arguments

client

Handle to slave device

Description

This executes the SMBus “receive byte” protocol, returning negative errno else the byte received from the device.

i2c_smbus_write_byte

LINUX

Name

`i2c_smbus_write_byte` — SMBus “send byte” protocol

Synopsis

```
s32 i2c_smbus_write_byte (const struct i2c_client * client, u8
value);
```

Arguments

client

Handle to slave device

value

Byte to be sent

Description

This executes the SMBus “send byte” protocol, returning negative errno else zero on success.

i2c_smbus_read_byte_data

LINUX

Name

`i2c_smbus_read_byte_data` — SMBus “read byte” protocol

Synopsis

```
s32 i2c_smbus_read_byte_data (const struct i2c_client *
client, u8 command);
```

Arguments

client

Handle to slave device

command

Byte interpreted by slave

Description

This executes the SMBus “read byte” protocol, returning negative errno else a data byte received from the device.

i2c_smbus_write_byte_data

LINUX

Name

`i2c_smbus_write_byte_data` — SMBus “write byte” protocol

Synopsis

```
s32 i2c_smbus_write_byte_data (const struct i2c_client *
client, u8 command, u8 value);
```

Arguments

client

Handle to slave device

command

Byte interpreted by slave

value

Byte being written

Description

This executes the SMBus “write byte” protocol, returning negative errno else zero on success.

i2c_smbus_read_word_data

LINUX

Name

`i2c_smbus_read_word_data` — SMBus “read word” protocol

Synopsis

```
s32 i2c_smbus_read_word_data (const struct i2c_client *
client, u8 command);
```

Arguments

client

Handle to slave device

command

Byte interpreted by slave

Description

This executes the SMBus “read word” protocol, returning negative errno else a 16-bit unsigned “word” received from the device.

i2c_smbus_write_word_data

LINUX

Name

`i2c_smbus_write_word_data` — SMBus “write word” protocol

Synopsis

```
s32 i2c_smbus_write_word_data (const struct i2c_client *
client, u8 command, u16 value);
```

Arguments

client

Handle to slave device

command

Byte interpreted by slave

value

16-bit “word” being written

Description

This executes the SMBus “write word” protocol, returning negative errno else zero on success.

i2c_smbus_process_call

LINUX

Name

`i2c_smbus_process_call` — SMBus “process call” protocol

Synopsis

```
s32 i2c_smbus_process_call (const struct i2c_client * client,
u8 command, u16 value);
```

Arguments

client

Handle to slave device

command

Byte interpreted by slave

value

16-bit “word” being written

Description

This executes the SMBus “process call” protocol, returning negative errno else a 16-bit unsigned “word” received from the device.

i2c_smbus_read_block_data

LINUX

Name

`i2c_smbus_read_block_data` — SMBus “block read” protocol

Synopsis

```
s32 i2c_smbus_read_block_data (const struct i2c_client *
client, u8 command, u8 * values);
```

Arguments

client

Handle to slave device

command

Byte interpreted by slave

values

Byte array into which data will be read; big enough to hold the data returned by the slave. SMBus allows at most 32 bytes.

Description

This executes the SMBus “block read” protocol, returning negative errno else the number of data bytes in the slave’s response.

Note that using this function requires that the client’s adapter support the I2C_FUNC_SMBUS_READ_BLOCK_DATA functionality. Not all adapter drivers support this; its emulation through I2C messaging relies on a specific mechanism (I2C_M_RECV_LEN) which may not be implemented.

i2c_smbus_write_block_data

LINUX

Kernel Hackers ManualApril 2016

Name

`i2c_smbus_write_block_data` — SMBus “block write” protocol

Synopsis

```
s32 i2c_smbus_write_block_data (const struct i2c_client *
client, u8 command, u8 length, const u8 * values);
```

Arguments

client

Handle to slave device

command

Byte interpreted by slave

length

Size of data block; SMBus allows at most 32 bytes

values

Byte array which will be written.

Description

This executes the SMBus “block write” protocol, returning negative errno else zero on success.

i2c_smbus_xfer

LINUX

Kernel Hackers ManualApril 2016

Name

`i2c_smbus_xfer` — execute SMBus protocol operations

Synopsis

```
s32 i2c_smbus_xfer (struct i2c_adapter * adapter, u16 addr,
unsigned short flags, char read_write, u8 command, int
protocol, union i2c_smbus_data * data);
```

Arguments

adapter

Handle to I2C bus

addr

Address of SMBus slave on that bus

flags

I2C_CLIENT_* flags (usually zero or I2C_CLIENT_PEC)

read_write

I2C_SMBUS_READ or I2C_SMBUS_WRITE

command

Byte interpreted by slave, for protocols which use such bytes

protocol

SMBus protocol operation to execute, such as I2C_SMBUS_PROC_CALL

`data`

Data to be read or written

Description

This executes an SMBus protocol operation, and returns a negative errno code else zero on success.