Linux Networking and Network Devices APIs

Linux Networking and Network Devices APIs

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. Linux Networking 1
1.1. Networking Base Types 1
enum sock_type 1
struct socket 2
1.2. Socket Buffer Functions 3
struct skb_shared_hwtstamps 3
struct sk_buff 5
skb_dst 7
skb_dst_set 8
skb_dst_is_noref 9
skb_queue_empty 10
skb_queue_is_last 11
skb_queue_is_first 11
skb_queue_next 12
skb_queue_prev 13
skb_get 14
skb_cloned 15
skb_header_cloned 16
skb_header_release 16
skb_shared 17
skb_share_check 18
skb_unshare 19
skb_peek 20
skb_peek_tail 21
skb_queue_len 22
__skb_queue_head_init 23
skb_queue_splice 23
skb_queue_splice_init 24
skb_queue_splice_tail 25
skb_queue_splice_tail_init 26
__skb_queue_after 27
__skb_fill_page_desc 28
skb_fill_page_desc 29
skb_headroom 30
skb_tailroom 31
skb_availroom 32
skb_reserve 32
skb_tailroom_reserve 33
pskb_trim_unique 34
skb_orphan 35
__dev_alloc_skb 36

netdev_alloc_skb 37
skb_frag_page 38
__skb_frag_ref 39
skb_frag_ref 40
__skb_frag_unref 40
skb_frag_unref 41
skb_frag_address 42
skb_frag_address_safe 43
__skb_frag_set_page 44
skb_frag_set_page 45
skb_frag_dma_map 45
skb_clone_writable 47
skb_cow 47
skb_cow_head 48
skb_padto 49
skb_put_padto 50
skb_linearize 51
__netdev_alloc_page 52
netdev_alloc_page 53
propagate_pfmemalloc_skb 54
skb_linearize_cow 55
skb_postpull_rcsum 56
pskb_trim_rcsum 57
skb_get_timestamp 57
skb_complete_tx_timestamp 58
skb_tx_timestamp 59
skb_checksum_complete 60
skb_checksum_none_assert 61
struct sock_common 62
struct sock 63
unlock_sock_fast 70
sk_filter_release 71
sk_wmem_alloc_get 72
sk_rmem_alloc_get 73
sk_has_allocations 74
wq_has_sleeper 74
sock_poll_wait 75
sk_eat_skb 76
sockfd_lookup 77
sock_release 78
kernel_recvmsg 79
sock_register 81
sock_unregister 81

__alloc_skb 82
__netdev_alloc_skb 84
dev_alloc_skb 84
__kfree_skb 85
kfree_skb 86
consume_skb 87
skb_recycle_check 88
skb_morph 89
skb_clone 90
skb_copy 91
pskb_copy 92
pskb_expand_head 93
skb_copy_expand 94
skb_pad 95
skb_put 96
skb_push 97
skb_pull 98
skb_pull_data 99
skb_trim 100
__pskb_pull_tail 101
skb_store_bits 102
skb_dequeue 103
skb_dequeue_tail 104
skb_queue_purge 105
skb_queue_head 105
skb_queue_tail 106
skb_unlink 107
skb_append 108
skb_insert 109
skb_split 110
skb_prepare_seq_read 111
skb_seq_read 112
skb_abort_seq_read 114
skb_find_text 115
skb_append_datato_frags 116
skb_pull_rcsum 117
skb_segment 118
skb_cow_data 119
skb_partial_csum_set 120
sk_set_memalloc 121
sk_alloc 122
sk_wait_data 123
__sk_mem_schedule 123

__sk_mem_reclaim 124
lock_sock_fast 125
__skb_recv_datagram 126
skb_kill_datagram 127
skb_copy_datagram_iovec 128
skb_copy_datagram_const_iovec 129
skb_copy_datagram_from_iovec 131
skb_copy_and_csum_datagram_iovec 132
datagram_poll 133
sk_stream_write_space 134
sk_stream_wait_connect 135
sk_stream_wait_memory 136
1.3. Socket Filter 136
sk_filter 137
sk_run_filter 137
sk_chk_filter 138
sk_filter_release_rcu 139
sk_attach_filter 140
1.4. Generic Network Statistics 141
struct gnet_stats_basic 141
struct gnet_stats_rate_est 142
struct gnet_stats_queue 143
struct gnet_estimator 144
gnet_stats_start_copy_compat 145
gnet_stats_start_copy 146
gnet_stats_copy_basic 147
gnet_stats_copy_rate_est 148
gnet_stats_copy_queue 149
gnet_stats_copy_app 150
gnet_stats_finish_copy 151
gen_new_estimator 152
gen_kill_estimator 153
gen_replace_estimator 154
gen_estimator_active 155
1.5. SUN RPC subsystem 156
xdr_encode_opaque_fixed 156
xdr_encode_opaque 157
xdr_terminate_string 158
xdr_init_encode 159
xdr_reserve_space 160
xdr_write_pages 161
xdr_init_decode 162
xdr_init_decode_pages 163

xdr_set_scratch_buffer 163
xdr_inline_decode 164
xdr_read_pages 165
xdr_enter_page 166
svc_print_addr 167
svc_reserve 168
svc_find_xprt 169
svc_xprt_names 170
xprt_register_transport 171
xprt_unregister_transport 172
xprt_load_transport 173
xprt_release_xprt 174
xprt_release_xprt_cong 175
xprt_release_rqst_cong 176
xprt_adjust_cwnd 176
xprt_wake_pending_tasks 177
xprt_wait_for_buffer_space 178
xprt_write_space 179
xprt_set_retrans_timeout_def 179
xprt_disconnect_done 180
xprt_lookup_rqst 181
xprt_complete_rqst 182
rpc_wake_up 183
rpc_wake_up_status 183
rpc_malloc 184
rpc_free 185
xdr_skb_read_bits 186
xdr_partial_copy_from_skb 187
csum_partial_copy_to_xdr 188
rpc_alloc_iostats 189
rpc_free_iostats 190
rpc_queue_upcall 190
rpc_mkpipe 191
rpc_unlink 193
rpcb_getport_async 194
rpc_bind_new_program 194
rpc_run_task 195
rpc_call_sync 196
rpc_call_async 197
rpc_peeraddr 198
rpc_peeraddr2str 199
rpc_force_rebind 200
1.6. WiMAX 200

wimax_msg_alloc 201
wimax_msg_data_len 202
wimax_msg_data 203
wimax_msg_len 204
wimax_msg_send 204
wimax_msg 206
wimax_reset 207
wimax_report_rfkill_hw 208
wimax_report_rfkill_sw 209
wimax_rfkill 210
wimax_state_change 212
wimax_state_get 213
wimax_dev_init 214
wimax_dev_add 214
wimax_dev_rm 215
struct wimax_dev 216
enum wimax_st 221
2. Network device support 225
2.1. Driver Support 225
dev_add_pack 225
__dev_remove_pack 225
dev_remove_pack 226
netdev_boot_setup_check 227
__dev_get_by_name 228
dev_get_by_name_rcu 229
dev_get_by_name 230
__dev_get_by_index 231
dev_get_by_index_rcu 232
dev_get_by_index 233
dev_getbyhwaddr_rcu 234
dev_get_by_flags_rcu 235
dev_valid_name 236
dev_alloc_name 237
netdev_features_change 238
netdev_state_change 238
dev_load 239
dev_open 240
dev_close 241
dev_disable_lro 242
register_netdevice_notifier 243
unregister_netdevice_notifier 244
call_netdevice_notifiers 245
dev_forward_skb 246

netif_set_real_num_rx_queues 247
netif_device_detach 247
netif_device_attach 248
skb_gso_segment 249
dev_queue_xmit 250
rps_may_expire_flow 251
netif_rx 252
netdev_rx_handler_register 253
netdev_rx_handler_unregister 254
netif_receive_skb 255
__napi_schedule 256
__napi_schedule_irqoff 257
register_gifconf 258
netdev_set_master 259
netdev_set_bond_master 260
dev_set_promiscuity 260
dev_set_allmulti 261
dev_ethtool_get_settings 262
dev_get_flags 264
dev_change_flags 264
dev_set_mtu 265
dev_set_group 266
dev_set_mac_address 267
dev_get_phys_port_id 268
netdev_update_features 268
netdev_change_features 269
netif_stacked_transfer_operstate 270
register_netdevice 271
init_dummy_netdev 272
register_netdev 273
dev_get_stats 274
alloc_netdev_mqs 275
free_netdev 276
synchronize_net 277
unregister_netdevice_queue 278
unregister_netdevice_many 279
unregister_netdev 279
dev_change_net_namespace 280
netdev_increment_features 281
eth_header 282
eth_rebuild_header 283
eth_type_trans 284
eth_header_parse 285

eth_header_cache 286
eth_header_cache_update 287
eth_mac_addr 288
eth_change_mtu 289
ether_setup 290
alloc_etherdev_mqs 290
netif_carrier_on 291
netif_carrier_off 292
netif_notify_peers 293
is_link_local_ether_addr 294
is_zero_ether_addr 295
is_multicast_ether_addr 295
is_local_ether_addr 296
is_broadcast_ether_addr 297
is_unicast_ether_addr 298
is_valid_ether_addr 299
eth_random_addr 300
eth_broadcast_addr 300
eth_zero_addr 301
eth_hw_addr_random 302
ether_addr_copy 303
compare_ether_addr 304
ether_addr_equal 304
compare_ether_addr_64bits 305
ether_addr_equal_unaligned 306
is_etherdev_addr 307
compare_ether_header 308
eth_skb_pad 309
napi_schedule_prep 310
napi_schedule 311
napi_schedule_irqoff 312
napi_disable 313
napi_enable 313
napi_synchronize 314
netdev_priv 315
netif_start_queue 316
netif_wake_queue 317
netif_stop_queue 317
netif_queue_stopped 318
netdev_txq_bql_enqueue_prefetchw 319
netdev_txq_bql_complete_prefetchw 320
netif_running 321
netif_start_subqueue 321

netif_stop_subqueue 322
__netif_subqueue_stopped 323
netif_wake_subqueue 324
netif_is_multiqueue 325
dev_put 326
dev_hold 327
netif_carrier_ok 327
netif_dormant_on 328
netif_dormant_off 329
netif_dormant 330
netif_oper_up 331
netif_device_present 331
netif_tx_lock 332
__dev_uc_sync 333
__dev_uc_unsync 334
__dev_mc_sync 335
__dev_mc_unsync 336
2.2. PHY Support 337
phy_print_status 337
phy_ethtool_sset 337
phy_mii_ioctl 338
phy_start_aneg 339
phy_start_interrupts 340
phy_stop_interrupts 341
phy_stop 342
phy_start 342
phy_clear_interrupt 343
phy_config_interrupt 344
phy_aneg_done 345
phy_find_setting 346
phy_find_valid 347
phy_sanitize_settings 347
phy_start_machine 348
phy_stop_machine 349
phy_force_reduction 350
phy_error 351
phy_interrupt 352
phy_enable_interrupts 353
phy_disable_interrupts 353
phy_change 354
phy_state_machine 355
get_phy_id 355
get_phy_device 356

phy_device_register 357
phy_find_first 358
phy_connect_direct 359
phy_connect 360
phy_disconnect 361
phy_attach 362
phy_detach 363
genphy_restart_aneg 363
genphy_config_aneg 364
genphy_update_link 365
genphy_read_status 366
phy_driver_register 367
phy_prepare_link 367
phy_attach_direct 368
genphy_config_advert 369
genphy_setup_forced 370
phy_probe 371
mdiobus_alloc 372
mdiobus_register 373
mdiobus_free 374
mdiobus_read 374
mdiobus_write 375
mdiobus_release 376
mdio_bus_match 377

Chapter 1. Linux Networking

1.1. Networking Base Types

enum sock_type

LINUX

Kernel Hackers ManualNovember 2026

Name

`enum sock_type` — Socket types

Synopsis

```
enum sock_type {
  SOCK_STREAM,
  SOCK_DGRAM,
  SOCK_RAW,
  SOCK_RDM,
  SOCK_SEQPACKET,
  SOCK_DCCP,
  SOCK_PACKET
};
```

Constants

SOCK_STREAM

stream (connection) socket

SOCK_DGRAM

datagram (conn.less) socket

SOCK_RAW

raw socket

SOCK_RDM

reliably-delivered message

SOCK_SEQPACKET

sequential packet socket

SOCK_DCCP

Datagram Congestion Control Protocol socket

SOCK_PACKET

linux specific way of getting packets at the dev level. For writing rarp and other similar things on the user level.

Description

When adding some new socket type please grep ARCH_HAS_SOCKET_TYPE include/asm-* /socket.h, at least MIPS overrides this enum for binary compat reasons.

struct socket

LINUX

Kernel Hackers ManualNovember 2026

Name

`struct socket` — general BSD socket

Synopsis

```
struct socket {
  socket_state state;
  short type;
  unsigned long flags;
  struct socket_wq __rcu * wq;
  struct file * file;
```

```
  struct sock * sk;
  const struct proto_ops * ops;
};
```

Members

state

socket state (`SS_CONNECTED`, etc)

type

socket type (`SOCK_STREAM`, etc)

flags

socket flags (`SOCK_ASYNC_NOSPACE`, etc)

wq

wait queue for several uses

file

File back pointer for gc

sk

internal networking protocol agnostic socket representation

ops

protocol specific socket operations

1.2. Socket Buffer Functions

struct skb_shared_hwtstamps

LINUX

Name

`struct skb_shared_hwtstamps` — hardware time stamps

Synopsis

```
struct skb_shared_hwtstamps {
  ktime_t hwtstamp;
  ktime_t syststamp;
};
```

Members

hwtstamp

hardware time stamp transformed into duration since arbitrary point in time

syststamp

hwtstamp transformed to system time base

Description

Software time stamps generated by `ktime_get_real` are stored in skb->tstamp. The relation between the different kinds of time

stamps is as follows

syststamp and tstamp can be compared against each other in arbitrary combinations. The accuracy of a syststamp/tstamp/“syststamp from other device” comparison is limited by the accuracy of the transformation into system time base. This depends on the device driver and its underlying hardware.

hwtstamps can only be compared against other hwtstamps from the same device.

This structure is attached to packets as part of the skb_shared_info. Use `skb_hwtstamps` to get a pointer.

struct sk_buff

LINUX

Kernel Hackers ManualNovember 2026

Name

`struct sk_buff` — socket buffer

Synopsis

```
struct sk_buff {
  struct sk_buff * next;
  struct sk_buff * prev;
  ktime_t tstamp;
  struct sock * sk;
  struct net_device * dev;
  char cb[48];
  unsigned long _skb_refdst;
#ifdef CONFIG_XFRM
  struct sec_path * sp;
#endif
  unsigned int len;
  unsigned int data_len;
  __u16 mac_len;
  __u16 hdr_len;
  union {unnamed_union};
  __u16 vlan_tci;
  sk_buff_data_t inner_transport_header;
  sk_buff_data_t inner_network_header;
  sk_buff_data_t transport_header;
  sk_buff_data_t network_header;
  sk_buff_data_t mac_header;
  sk_buff_data_t tail;
  sk_buff_data_t end;
  unsigned char * head;
  unsigned char * data;
  unsigned int truesize;
  atomic_t users;
};
```

Members

next

Next buffer in list

prev

Previous buffer in list

tstamp

Time we arrived

sk

Socket we are owned by

dev

Device we arrived on/are leaving by

cb[48]

Control buffer. Free for use by every layer. Put private vars here

_skb_refdst

destination entry (with norefcount bit)

sp

the security path, used for xfrm

len

Length of actual data

data_len

Data length

mac_len

Length of link layer header

hdr_len

writable header length of cloned skb

{unnamed_union}

anonymous

vlan_tci

: vlan tag control information

inner_transport_header

: Inner transport layer header (encapsulation)

inner_network_header

: Network layer header (encapsulation)

transport_header

: Transport layer header

network_header

: Network layer header

mac_header

: Link layer header

tail

: Tail pointer

end

: End pointer

head

: Head of buffer

data

: Data head pointer

truesize

: Buffer size

users

: User count - see {datagram,tcp}.c

skb_dst

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_dst — returns skb dst_entry

Synopsis

```
struct dst_entry * skb_dst (const struct sk_buff * skb);
```

Arguments

skb

buffer

Description

Returns skb dst_entry, regardless of reference taken or not.

skb_dst_set

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_dst_set — sets skb dst

Synopsis

```
void skb_dst_set (struct sk_buff * skb, struct dst_entry *
dst);
```

Arguments

skb

buffer

dst

dst entry

Description

Sets skb dst, assuming a reference was taken on dst and should be released by `skb_dst_drop`

skb_dst_is_noref

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_dst_is_noref` — Test if skb dst isn't refcounted

Synopsis

```
bool skb_dst_is_noref (const struct sk_buff * skb);
```

Arguments

skb

buffer

skb_queue_empty

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_queue_empty` — check if a queue is empty

Synopsis

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

Arguments

list

queue head

Description

Returns true if the queue is empty, false otherwise.

skb_queue_is_last

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_queue_is_last` — check if skb is the last entry in the queue

Synopsis

```
bool skb_queue_is_last (const struct sk_buff_head * list,
const struct sk_buff * skb);
```

Arguments

list

queue head

skb

buffer

Description

Returns true if *skb* is the last buffer on the list.

skb_queue_is_first

LINUX

Name

`skb_queue_is_first` — check if skb is the first entry in the queue

Synopsis

```
bool skb_queue_is_first (const struct sk_buff_head * list,
const struct sk_buff * skb);
```

Arguments

`list`

queue head

`skb`

buffer

Description

Returns true if *`skb`* is the first buffer on the list.

skb_queue_next

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_queue_next` — return the next packet in the queue

Synopsis

```
struct sk_buff * skb_queue_next (const struct sk_buff_head *
list, const struct sk_buff * skb);
```

Arguments

list

queue head

skb

current buffer

Description

Return the next packet in *list* after *skb*. It is only valid to call this if skb_queue_is_last evaluates to false.

skb_queue_prev

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_queue_prev — return the prev packet in the queue

Synopsis

```
struct sk_buff * skb_queue_prev (const struct sk_buff_head *
list, const struct sk_buff * skb);
```

Arguments

list

queue head

skb

current buffer

Description

Return the prev packet in *list* before *skb*. It is only valid to call this if skb_queue_is_first evaluates to false.

skb_get

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_get — reference buffer

Synopsis

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

Arguments

skb

buffer to reference

Description

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

skb_cloned

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_cloned — is the buffer a clone

Synopsis

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

Arguments

skb

buffer to check

Description

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

skb_header_cloned

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_header_cloned` — is the header a clone

Synopsis

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

Arguments

`skb`

buffer to check

Description

Returns true if modifying the header part of the buffer requires the data to be copied.

skb_header_release

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_header_release — release reference to header

Synopsis

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

Arguments

skb

buffer to operate on

Description

Drop a reference to the header part of the buffer. This is done by acquiring a payload reference. You must not read from the header part of skb->data after this.

skb_shared

LINUX

Name

`skb_shared` — is the buffer shared

Synopsis

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

Arguments

`skb`

buffer to check

Description

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

skb_share_check

LINUX

Name

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

Synopsis

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

Arguments

skb

buffer to check

pri

priority for memory allocation

Description

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

NULL is returned on a memory allocation failure.

skb_unshare

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_unshare` — make a copy of a shared buffer

Synopsis

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

Arguments

skb

buffer to check

pri

priority for memory allocation

Description

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

NULL is returned on a memory allocation failure.

skb_peek

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_peek — peek at the head of an sk_buff_head

Synopsis

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

Arguments

*list_*

list to peek at

Description

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

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

skb_peek_tail

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_peek_tail — peek at the tail of an sk_buff_head

Synopsis

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

Arguments

*list_*

list to peek at

Description

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

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

skb_queue_len

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_queue_len` — get queue length

Synopsis

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

Arguments

*list_*

list to measure

Description

Return the length of an sk_buff queue.

__skb_queue_head_init

LINUX

Kernel Hackers ManualNovember 2026

Name

`__skb_queue_head_init` — initialize non-spinlock portions of sk_buff_head

Synopsis

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

Arguments

list

queue to initialize

Description

This initializes only the list and queue length aspects of an sk_buff_head object. This allows to initialize the list aspects of an sk_buff_head without reinitializing things like the spinlock. It can also be used for on-stack sk_buff_head objects where the spinlock is known to not be used.

skb_queue_splice

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_queue_splice` — join two skb lists, this is designed for stacks

Synopsis

```
void skb_queue_splice (const struct sk_buff_head * list,
struct sk_buff_head * head);
```

Arguments

`list`

the new list to add

`head`

the place to add it in the first list

skb_queue_splice_init

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_queue_splice_init` — join two skb lists and reinitialise the emptied list

Synopsis

```
void skb_queue_splice_init (struct sk_buff_head * list, struct
sk_buff_head * head);
```

Arguments

list

the new list to add

head

the place to add it in the first list

Description

The list at *list* is reinitialised

skb_queue_splice_tail

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_queue_splice_tail — join two skb lists, each list being a queue

Synopsis

```
void skb_queue_splice_tail (const struct sk_buff_head * list,
struct sk_buff_head * head);
```

Arguments

list

the new list to add

head

the place to add it in the first list

skb_queue_splice_tail_init

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_queue_splice_tail_init` — join two skb lists and reinitialise the emptied list

Synopsis

```
void skb_queue_splice_tail_init (struct sk_buff_head * list,
struct sk_buff_head * head);
```

Arguments

list

the new list to add

head

the place to add it in the first list

Description

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

__skb_queue_after

LINUX

Kernel Hackers ManualNovember 2026

Name

__skb_queue_after — queue a buffer at the list head

Synopsis

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

Arguments

list

list to use

prev

place after this buffer

newsk

buffer to queue

Description

Queue a buffer int the middle of a list. This function takes no locks and you must therefore hold required locks before calling it.

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

__skb_fill_page_desc

LINUX

Kernel Hackers ManualNovember 2026

Name

`__skb_fill_page_desc` — initialise a paged fragment in an skb

Synopsis

```
void __skb_fill_page_desc (struct sk_buff * skb, int i, struct
page * page, int off, int size);
```

Arguments

skb

buffer containing fragment to be initialised

i

paged fragment index to initialise

page

the page to use for this fragment

off

the offset to the data with *page*

size

the length of the data

Description

Initialises the *i*'th fragment of *skb* to point to size bytes at offset *off* within *page*.

Does not take any additional reference on the fragment.

skb_fill_page_desc

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_fill_page_desc — initialise a paged fragment in an skb

Synopsis

```
void skb_fill_page_desc (struct sk_buff * skb, int i, struct
page * page, int off, int size);
```

Arguments

skb

buffer containing fragment to be initialised

i

paged fragment index to initialise

page

the page to use for this fragment

off

the offset to the data with *page*

size

the length of the data

Description

As per __skb_fill_page_desc -- initialises the *i*'th fragment of *skb* to point to size bytes at offset *off* within *page*. In addition updates *skb* such that *i* is the last fragment.

Does not take any additional reference on the fragment.

skb_headroom

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_headroom — bytes at buffer head

Synopsis

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

Arguments

skb

buffer to check

Description

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

skb_tailroom

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_tailroom — bytes at buffer end

Synopsis

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

Arguments

skb

buffer to check

Description

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

skb_availroom

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_availroom` — bytes at buffer end

Synopsis

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

Arguments

skb

buffer to check

Description

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

skb_reserve

LINUX

Name

`skb_reserve` — adjust headroom

Synopsis

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

Arguments

skb

buffer to alter

len

bytes to move

Description

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

skb_tailroom_reserve

LINUX

Name

`skb_tailroom_reserve` — adjust reserved_tailroom

Synopsis

```
void skb_tailroom_reserve (struct sk_buff * skb, unsigned int
mtu, unsigned int needed_tailroom);
```

Arguments

skb

buffer to alter

mtu

maximum amount of headlen permitted

needed_tailroom

minimum amount of reserved_tailroom

Description

Set reserved_tailroom so that headlen can be as large as possible but not larger than mtu and tailroom cannot be smaller than needed_tailroom. The required headroom should already have been reserved before using this function.

pskb_trim_unique

LINUX

Kernel Hackers ManualNovember 2026

Name

`pskb_trim_unique` — remove end from a paged unique (not cloned) buffer

Synopsis

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

Arguments

skb

buffer to alter

len

new length

Description

This is identical to pskb_trim except that the caller knows that the skb is not cloned so we should never get an error due to out- of-memory.

skb_orphan

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_orphan — orphan a buffer

Synopsis

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

Arguments

skb

buffer to orphan

Description

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

__dev_alloc_skb

LINUX

Kernel Hackers ManualNovember 2026

Name

__dev_alloc_skb — allocate an skbuff for receiving

Synopsis

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

Arguments

length

length to allocate

gfp_mask

get_free_pages mask, passed to alloc_skb

Description

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

NULL is returned if there is no free memory.

netdev_alloc_skb

LINUX

Kernel Hackers ManualNovember 2026

Name

netdev_alloc_skb — allocate an skbuff for rx on a specific device

Synopsis

```
struct sk_buff * netdev_alloc_skb (struct net_device * dev,
unsigned int length);
```

Arguments

dev

network device to receive on

length

length to allocate

Description

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

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

skb_frag_page

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_frag_page — retrieve the page refered to by a paged fragment

Synopsis

```
struct page * skb_frag_page (const skb_frag_t * frag);
```

Arguments

frag

the paged fragment

Description

Returns the struct page associated with *frag*.

__skb_frag_ref

LINUX

Kernel Hackers ManualNovember 2026

Name

__skb_frag_ref — take an addition reference on a paged fragment.

Synopsis

```
void __skb_frag_ref (skb_frag_t * frag);
```

Arguments

frag

the paged fragment

Description

Takes an additional reference on the paged fragment *frag*.

skb_frag_ref

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_frag_ref` — take an addition reference on a paged fragment of an skb.

Synopsis

```
void skb_frag_ref (struct sk_buff * skb, int f);
```

Arguments

skb

the buffer

f

the fragment offset.

Description

Takes an additional reference on the *f*'th paged fragment of *skb*.

__skb_frag_unref

LINUX

Kernel Hackers ManualNovember 2026

Name

__skb_frag_unref — release a reference on a paged fragment.

Synopsis

```
void __skb_frag_unref (skb_frag_t * frag);
```

Arguments

frag

the paged fragment

Description

Releases a reference on the paged fragment *frag*.

skb_frag_unref

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_frag_unref — release a reference on a paged fragment of an skb.

Synopsis

```
void skb_frag_unref (struct sk_buff * skb, int f);
```

Arguments

skb

the buffer

f

the fragment offset

Description

Releases a reference on the *f*'th paged fragment of *skb*.

skb_frag_address

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_frag_address — gets the address of the data contained in a paged fragment

Synopsis

```
void * skb_frag_address (const skb_frag_t * frag);
```

Arguments

frag

the paged fragment buffer

Description

Returns the address of the data within *frag*. The page must already be mapped.

skb_frag_address_safe

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_frag_address_safe — gets the address of the data contained in a paged fragment

Synopsis

```
void * skb_frag_address_safe (const skb_frag_t * frag);
```

Arguments

frag

the paged fragment buffer

Description

Returns the address of the data within *frag*. Checks that the page is mapped and returns NULL otherwise.

__skb_frag_set_page

LINUX

Kernel Hackers ManualNovember 2026

Name

__skb_frag_set_page — sets the page contained in a paged fragment

Synopsis

```
void __skb_frag_set_page (skb_frag_t * frag, struct page *
page);
```

Arguments

frag

the paged fragment

page

the page to set

Description

Sets the fragment *frag* to contain *page*.

skb_frag_set_page

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_frag_set_page` — sets the page contained in a paged fragment of an skb

Synopsis

```
void skb_frag_set_page (struct sk_buff * skb, int f, struct
page * page);
```

Arguments

skb

the buffer

f

the fragment offset

page

the page to set

Description

Sets the *f*'th fragment of *skb* to contain *page*.

skb_frag_dma_map

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_frag_dma_map` — maps a paged fragment via the DMA API

Synopsis

```
dma_addr_t skb_frag_dma_map (struct device * dev, const
skb_frag_t * frag, size_t offset, size_t size, enum
dma_data_direction dir);
```

Arguments

`dev`

-- undescribed --

`frag`

the paged fragment to map

`offset`

the offset within the fragment (starting at the fragment's own offset)

`size`

the number of bytes to map

`dir`

-- undescribed --

Description

Maps the page associated with *`frag`* to *`device`*.

skb_clone_writable

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_clone_writable` — is the header of a clone writable

Synopsis

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

Arguments

skb

buffer to check

len

length up to which to write

Description

Returns true if modifying the header part of the cloned buffer does not requires the data to be copied.

skb_cow

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_cow — copy header of skb when it is required

Synopsis

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

Arguments

skb

buffer to cow

headroom

needed headroom

Description

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

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

skb_cow_head

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_cow_head` — skb_cow but only making the head writable

Synopsis

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

Arguments

skb

buffer to cow

headroom

needed headroom

Description

This function is identical to skb_cow except that we replace the skb_cloned check by skb_header_cloned. It should be used when you only need to push on some header and do not need to modify the data.

skb_padto

LINUX

Name

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

Synopsis

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

Arguments

skb

buffer to pad

len

minimal length

Description

Pads up a buffer to ensure the trailing bytes exist and are blanked. If the buffer already contains sufficient data it is untouched. Otherwise it is extended. Returns zero on success. The skb is freed on error.

skb_put_padto

LINUX

Name

`skb_put_padto` — increase size and pad an skbuff up to a minimal size

Synopsis

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

Arguments

skb

buffer to pad

len

minimal length

Description

Pads up a buffer to ensure the trailing bytes exist and are blanked. If the buffer already contains sufficient data it is untouched. Otherwise it is extended. Returns zero on success. The skb is freed on error.

skb_linearize

LINUX

Name

`skb_linearize` — convert paged skb to linear one

Synopsis

```
int skb_linearize (struct sk_buff * skb);
```

Arguments

skb

buffer to linarize

Description

If there is no free memory -ENOMEM is returned, otherwise zero is returned and the old skb data released.

__netdev_alloc_page

LINUX

Name

`__netdev_alloc_page` — allocate a page for ps-rx on a specific device

Synopsis

```
struct page * __netdev_alloc_page (struct net_device * dev,
gfp_t gfp_mask, struct sk_buff * skb);
```

Arguments

dev

-- undescribed --

gfp_mask

alloc_pages_node mask. Set __GFP_NOMEMALLOC if not for network packet RX

skb

skb to set pfmemalloc on if __GFP_MEMALLOC is used

Description

Allocate a new page. dev currently unused.

`NULL` is returned if there is no free memory.

netdev_alloc_page

LINUX

Kernel Hackers ManualNovember 2026

Name

`netdev_alloc_page` — allocate a page for ps-rx on a specific device

Synopsis

```
struct page * netdev_alloc_page (struct net_device * dev);
```

Arguments

dev

network device to receive on

Description

Allocate a new page. dev currently unused.

`NULL` is returned if there is no free memory.

propagate_pfmemalloc_skb

LINUX

Kernel Hackers ManualNovember 2026

Name

`propagate_pfmemalloc_skb` — Propagate pfmemalloc if skb is allocated after RX page

Synopsis

```
void propagate_pfmemalloc_skb (struct page * page, struct
sk_buff * skb);
```

Arguments

page

The page that was allocated from netdev_alloc_page

skb

The skb that may need pfmemalloc set

skb_linearize_cow

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_linearize_cow` — make sure skb is linear and writable

Synopsis

```
int skb_linearize_cow (struct sk_buff * skb);
```

Arguments

skb

buffer to process

Description

If there is no free memory -ENOMEM is returned, otherwise zero is returned and the old skb data released.

skb_postpull_rcsum

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_postpull_rcsum` — update checksum for received skb after pull

Synopsis

```
void skb_postpull_rcsum (struct sk_buff * skb, const void *
start, unsigned int len);
```

Arguments

skb

buffer to update

start

start of data before pull

len

length of data pulled

Description

After doing a pull on a received packet, you need to call this to update the CHECKSUM_COMPLETE checksum, or set ip_summed to CHECKSUM_NONE so that it can be recomputed from scratch.

pskb_trim_rcsum

LINUX

Kernel Hackers ManualNovember 2026

Name

`pskb_trim_rcsum` — trim received skb and update checksum

Synopsis

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

Arguments

skb

buffer to trim

len

new length

Description

This is exactly the same as pskb_trim except that it ensures the checksum of received packets are still valid after the operation.

skb_get_timestamp

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_get_timestamp — get timestamp from a skb

Synopsis

```
void skb_get_timestamp (const struct sk_buff * skb, struct
timeval * stamp);
```

Arguments

skb

skb to get stamp from

stamp

pointer to struct timeval to store stamp in

Description

Timestamps are stored in the skb as offsets to a base timestamp. This function converts the offset back to a struct timeval and stores it in stamp.

skb_complete_tx_timestamp

LINUX

Name

`skb_complete_tx_timestamp` — deliver cloned skb with tx timestamps

Synopsis

```
void skb_complete_tx_timestamp (struct sk_buff * skb, struct
skb_shared_hwtstamps * hwtstamps);
```

Arguments

skb

clone of the the original outgoing packet

hwtstamps

hardware time stamps, may be NULL if not available

Description

PHY drivers may accept clones of transmitted packets for timestamping via their phy_driver.txtstamp method. These drivers must call this function to return the skb back to the stack, with or without a timestamp.

skb_tx_timestamp

LINUX

Name

`skb_tx_timestamp` — Driver hook for transmit timestamping

Synopsis

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

Arguments

`skb`

A socket buffer.

Description

Ethernet MAC Drivers should call this function in their `hard_xmit` function as soon as possible after giving the sk_buff to the MAC hardware, but before freeing the sk_buff.

skb_checksum_complete

LINUX

Name

`skb_checksum_complete` — Calculate checksum of an entire packet

Synopsis

```
__sum16 skb_checksum_complete (struct sk_buff * skb);
```

Arguments

skb

packet to process

Description

This function calculates the checksum over the entire packet plus the value of skb->csum. The latter can be used to supply the checksum of a pseudo header as used by TCP/UDP. It returns the checksum.

For protocols that contain complete checksums such as ICMP/TCP/UDP, this function can be used to verify that checksum on received packets. In that case the function should return zero if the checksum is correct. In particular, this function will return zero if skb->ip_summed is CHECKSUM_UNNECESSARY which indicates that the hardware has already verified the correctness of the checksum.

skb_checksum_none_assert

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_checksum_none_assert` — make sure skb ip_summed is CHECKSUM_NONE

Synopsis

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

Arguments

skb

skb to check

Description

fresh skbs have their ip_summed set to CHECKSUM_NONE. Instead of forcing ip_summed to CHECKSUM_NONE, we can use this helper, to document places where we make this assertion.

struct sock_common

LINUX

Kernel Hackers ManualNovember 2026

Name

`struct sock_common` — minimal network layer representation of sockets

Synopsis

```
struct sock_common {
  __be32 skc_daddr;
  __be32 skc_rcv_saddr;
  union {unnamed_union};
  int skc_tx_queue_mapping;
  atomic_t skc_refcnt;
```

```
};
```

Members

skc_daddr

Foreign IPv4 addr

skc_rcv_saddr

Bound local IPv4 addr

{unnamed_union}

anonymous

skc_tx_queue_mapping

tx queue number for this connection

skc_refcnt

reference count

Description

This is the minimal network layer representation of sockets, the header for struct sock and struct inet_timewait_sock.

struct sock

LINUX

Kernel Hackers ManualNovember 2026

Name

`struct sock` — network layer representation of sockets

Synopsis

```
struct sock {
  struct sock_common __sk_common;
#define sk_node   __sk_common.skc_node
#define sk_nulls_node  __sk_common.skc_nulls_node
#define sk_refcnt  __sk_common.skc_refcnt
#define sk_tx_queue_mapping __sk_common.skc_tx_queue_mapping
#define sk_dontcopy_begin __sk_common.skc_dontcopy_begin
#define sk_dontcopy_end  __sk_common.skc_dontcopy_end
#define sk_hash   __sk_common.skc_hash
#define sk_family  __sk_common.skc_family
#define sk_state  __sk_common.skc_state
#define sk_reuse  __sk_common.skc_reuse
#define sk_bound_dev_if  __sk_common.skc_bound_dev_if
#define sk_bind_node  __sk_common.skc_bind_node
#define sk_prot   __sk_common.skc_prot
#define sk_net   __sk_common.skc_net
  socket_lock_t sk_lock;
  struct sk_buff_head sk_receive_queue;
  struct sk_backlog;
#define sk_rmem_alloc sk_backlog.rmem_alloc
  int sk_forward_alloc;
#ifdef CONFIG_RPS
  __u32 sk_rxhash;
#endif
  atomic_t sk_drops;
  int sk_rcvbuf;
  struct sk_filter __rcu * sk_filter;
  struct socket_wq __rcu * sk_wq;
#ifdef CONFIG_NET_DMA
#ifdef __GENKSYMS__
  struct sk_buff_head sk_async_wait_queue;
#else
#endif
#endif
#ifdef CONFIG_XFRM
  struct xfrm_policy * sk_policy[2];
#endif
  unsigned long sk_flags;
  struct dst_entry * sk_dst_cache;
  spinlock_t sk_dst_lock;
  atomic_t sk_wmem_alloc;
  atomic_t sk_omem_alloc;
  int sk_sndbuf;
  struct sk_buff_head sk_write_queue;
  unsigned int sk_shutdown:2;
  unsigned int sk_no_check:2;
```

```
  unsigned int sk_userlocks:4;
  unsigned int sk_protocol:8;
#define SK_PROTOCOL_MAX ((u8)~0U)
  int sk_wmem_queued;
  gfp_t sk_allocation;
  int sk_route_caps;
  int sk_route_nocaps;
  int sk_gso_type;
  unsigned int sk_gso_max_size;
  u16 sk_gso_max_segs;
  int sk_rcvlowat;
  unsigned long sk_lingertime;
  struct sk_buff_head sk_error_queue;
  struct proto * sk_prot_creator;
  rwlock_t sk_callback_lock;
  int sk_err;
  int sk_err_soft;
  unsigned short sk_ack_backlog;
  unsigned short sk_max_ack_backlog;
  __u32 sk_priority;
  struct pid * sk_peer_pid;
  const struct cred * sk_peer_cred;
  long sk_rcvtimeo;
  long sk_sndtimeo;
  void * sk_protinfo;
  struct timer_list sk_timer;
  ktime_t sk_stamp;
  struct socket * sk_socket;
  void * sk_user_data;
  struct page * sk_sndmsg_page;
  struct sk_buff * sk_send_head;
  __u32 sk_sndmsg_off;
  int sk_write_pending;
#ifdef CONFIG_SECURITY
  void * sk_security;
#endif
  __u32 sk_mark;
  u32 sk_classid;
  void (* sk_state_change) (struct sock *sk);
  void (* sk_data_ready) (struct sock *sk, int bytes);
  void (* sk_write_space) (struct sock *sk);
  void (* sk_error_report) (struct sock *sk);
  int (* sk_backlog_rcv) (struct sock *sk,struct sk_buff *skb);
  void (* sk_destruct) (struct sock *sk);
};
```

Members

__sk_common

shared layout with inet_timewait_sock

sk_lock

synchronizer

sk_receive_queue

incoming packets

sk_backlog

always used with the per-socket spinlock held

sk_forward_alloc

space allocated forward

sk_rxhash

flow hash received from netif layer

sk_drops

raw/udp drops counter

sk_rcvbuf

size of receive buffer in bytes

sk_filter

socket filtering instructions

sk_wq

sock wait queue and async head

sk_async_wait_queue

DMA copied packets

sk_policy[2]

flow policy

sk_flags

: `SO_LINGER` (l_onoff), `SO_BROADCAST`, `SO_KEEPALIVE`, `SO_OOBINLINE` settings, `SO_TIMESTAMPING` settings

sk_dst_cache

: destination cache

sk_dst_lock

: destination cache lock

sk_wmem_alloc

: transmit queue bytes committed

sk_omem_alloc

: "o“ is ”option“ or ”other"

sk_sndbuf

: size of send buffer in bytes

sk_write_queue

: Packet sending queue

sk_shutdown

: mask of `SEND_SHUTDOWN` and/or `RCV_SHUTDOWN`

sk_no_check

: `SO_NO_CHECK` setting, wether or not checkup packets

sk_userlocks

: `SO_SNDBUF` and `SO_RCVBUF` settings

sk_protocol

: which protocol this socket belongs in this network family

sk_wmem_queued

: persistent queue size

sk_allocation

: allocation mode

sk_route_caps

route capabilities (e.g. `NETIF_F_TSO`)

sk_route_nocaps

forbidden route capabilities (e.g NETIF_F_GSO_MASK)

sk_gso_type

GSO type (e.g. `SKB_GSO_TCPV4`)

sk_gso_max_size

Maximum GSO segment size to build

sk_gso_max_segs

Maximum number of GSO segments

sk_rcvlowat

`SO_RCVLOWAT` setting

sk_lingertime

`SO_LINGER` l_linger setting

sk_error_queue

rarely used

sk_prot_creator

sk_prot of original sock creator (see ipv6_setsockopt, IPV6_ADDRFORM for instance)

sk_callback_lock

used with the callbacks in the end of this struct

sk_err

last error

sk_err_soft

errors that don't cause failure but are the cause of a persistent failure not just 'timed out'

sk_ack_backlog

current listen backlog

sk_max_ack_backlog

: listen backlog set in `listen`

sk_priority

: `SO_PRIORITY` setting

sk_peer_pid

: struct pid for this socket's peer

sk_peer_cred

: `SO_PEERCRED` setting

sk_rcvtimeo

: `SO_RCVTIMEO` setting

sk_sndtimeo

: `SO_SNDTIMEO` setting

sk_protinfo

: private area, net family specific, when not using slab

sk_timer

: sock cleanup timer

sk_stamp

: time stamp of last packet received

sk_socket

: Identd and reporting IO signals

sk_user_data

: RPC layer private data

sk_sndmsg_page

: cached page for sendmsg

sk_send_head

: front of stuff to transmit

sk_sndmsg_off

: cached offset for sendmsg

sk_write_pending

: a write to stream socket waits to start

sk_security

: used by security modules

sk_mark

: generic packet mark

sk_classid

: this socket's cgroup classid

sk_state_change

: callback to indicate change in the state of the sock

sk_data_ready

: callback to indicate there is data to be processed

sk_write_space

: callback to indicate there is bf sending space available

sk_error_report

: callback to indicate errors (e.g. `MSG_ERRQUEUE`)

sk_backlog_rcv

: callback to process the backlog

sk_destruct

: called at sock freeing time, i.e. when all refcnt == 0

unlock_sock_fast

LINUX

Name

`unlock_sock_fast` — complement of lock_sock_fast

Synopsis

```
void unlock_sock_fast (struct sock * sk, bool slow);
```

Arguments

sk

socket

slow

slow mode

Description

fast unlock socket for user context. If slow mode is on, we call regular `release_sock`

sk_filter_release

LINUX

Name

`sk_filter_release` — release a socket filter

Synopsis

```
void sk_filter_release (struct sk_filter * fp);
```

Arguments

fp

filter to remove

Description

Remove a filter from a socket and release its resources.

sk_wmem_alloc_get

LINUX

Kernel Hackers ManualNovember 2026

Name

`sk_wmem_alloc_get` — returns write allocations

Synopsis

```
int sk_wmem_alloc_get (const struct sock * sk);
```

Arguments

sk

socket

Description

Returns sk_wmem_alloc minus initial offset of one

sk_rmem_alloc_get

LINUX

Kernel Hackers ManualNovember 2026

Name

`sk_rmem_alloc_get` — returns read allocations

Synopsis

```
int sk_rmem_alloc_get (const struct sock * sk);
```

Arguments

sk

socket

Description

Returns sk_rmem_alloc

sk_has_allocations

LINUX

Kernel Hackers ManualNovember 2026

Name

`sk_has_allocations` — check if allocations are outstanding

Synopsis

```
int sk_has_allocations (const struct sock * sk);
```

Arguments

sk

socket

Description

Returns true if socket has write or read allocations

wq_has_sleeper

LINUX

Name

`wq_has_sleeper` — check if there are any waiting processes

Synopsis

```
bool wq_has_sleeper (struct socket_wq * wq);
```

Arguments

wq

struct socket_wq

Description

Returns true if socket_wq has waiting processes

The purpose of the wq_has_sleeper and sock_poll_wait is to wrap the memory barrier call. They were added due to the race found within the tcp code.

Consider following tcp code paths

CPU1 CPU2

sys_select receive packet ... ... __add_wait_queue update tp->rcv_nxt ... ... tp->rcv_nxt check sock_def_readable ... { schedule `rcu_read_lock`; wq = rcu_dereference(sk->sk_wq); if (wq && waitqueue_active(wq->wait)) wake_up_interruptible(wq->wait) ... }

The race for tcp fires when the __add_wait_queue changes done by CPU1 stay in its cache, and so does the tp->rcv_nxt update on CPU2 side. The CPU1 could then endup calling schedule and sleep forever if there are no more data on the socket.

sock_poll_wait

LINUX

Kernel Hackers ManualNovember 2026

Name

`sock_poll_wait` — place memory barrier behind the poll_wait call.

Synopsis

```
void sock_poll_wait (struct file * filp, wait_queue_head_t *
wait_address, poll_table * p);
```

Arguments

filp

file

wait_address

socket wait queue

p

poll_table

Description

See the comments in the wq_has_sleeper function.

sk_eat_skb

LINUX

Kernel Hackers ManualNovember 2026

Name

`sk_eat_skb` — Release a skb if it is no longer needed

Synopsis

```
void sk_eat_skb (struct sock * sk, struct sk_buff * skb, int
copied_early);
```

Arguments

sk

socket to eat this skb from

skb

socket buffer to eat

copied_early

flag indicating whether DMA operations copied this data early

Description

This routine must be called with interrupts disabled or with the socket locked so that the sk_buff queue operation is ok.

sockfd_lookup

LINUX

Kernel Hackers ManualNovember 2026

Name

`sockfd_lookup` — Go from a file number to its socket slot

Synopsis

```
struct socket * sockfd_lookup (int fd, int * err);
```

Arguments

fd

file handle

err

pointer to an error code return

Description

The file handle passed in is locked and the socket it is bound too is returned. If an error occurs the err pointer is overwritten with a negative errno code and NULL is returned. The function checks for both invalid handles and passing a handle which is not a socket.

On a success the socket object pointer is returned.

sock_release

LINUX

Kernel Hackers ManualNovember 2026

Name

`sock_release` — close a socket

Synopsis

```
void sock_release (struct socket * sock);
```

Arguments

sock

socket to close

Description

The socket is released from the protocol stack if it has a release callback, and the inode is then released if the socket is bound to an inode not a file.

kernel_recvmsg

LINUX

Name

`kernel_recvmsg` — Receive a message from a socket (kernel space)

Synopsis

```
int kernel_recvmsg (struct socket * sock, struct msghdr * msg,
struct kvec * vec, size_t num, size_t size, int flags);
```

Arguments

`sock`

The socket to receive the message from

`msg`

Received message

`vec`

Input s/g array for message data

`num`

Size of input s/g array

`size`

Number of bytes to read

`flags`

Message flags (MSG_DONTWAIT, etc...)

Description

On return the msg structure contains the scatter/gather array passed in the vec argument. The array is modified so that it consists of the unfilled portion of the original array.

The returned value is the total number of bytes received, or an error.

sock_register

LINUX

Kernel Hackers ManualNovember 2026

Name

`sock_register` — add a socket protocol handler

Synopsis

```
int sock_register (const struct net_proto_family * ops);
```

Arguments

ops

description of protocol

Description

This function is called by a protocol handler that wants to advertise its address family, and have it linked into the socket interface. The value ops->family coresponds to the socket system call protocol family.

sock_unregister

LINUX

Kernel Hackers ManualNovember 2026

Name

`sock_unregister` — remove a protocol handler

Synopsis

```
void sock_unregister (int family);
```

Arguments

`family`

protocol family to remove

Description

This function is called by a protocol handler that wants to remove its address family, and have it unlinked from the new socket creation.

If protocol handler is a module, then it can use module reference counts to protect against new references. If protocol handler is not a module then it needs to provide its own protection in the ops->create routine.

__alloc_skb

LINUX

Name

__alloc_skb — allocate a network buffer

Synopsis

```
struct sk_buff * __alloc_skb (unsigned int size, gfp_t
gfp_mask, int flags, int node);
```

Arguments

size

size to allocate

gfp_mask

allocation mask

flags

If SKB_ALLOC_FCLONE is set, allocate from fclone cache instead of head cache and allocate a cloned (child) skb. If SKB_ALLOC_RX is set, __GFP_MEMALLOC will be used for allocations in case the data is required for writeback

node

numa node to allocate memory on

Description

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

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

__netdev_alloc_skb

LINUX

Kernel Hackers ManualNovember 2026

Name

__netdev_alloc_skb — allocate an skbuff for rx on a specific device

Synopsis

```
struct sk_buff * __netdev_alloc_skb (struct net_device * dev,
unsigned int length, gfp_t gfp_mask);
```

Arguments

dev

network device to receive on

length

length to allocate

gfp_mask

get_free_pages mask, passed to alloc_skb

Description

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

NULL is returned if there is no free memory.

dev_alloc_skb

LINUX

Kernel Hackers ManualNovember 2026

Name

`dev_alloc_skb` — allocate an skbuff for receiving

Synopsis

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

Arguments

`length`

length to allocate

Description

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

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

__kfree_skb

LINUX

Name

`__kfree_skb` — private function

Synopsis

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

Arguments

skb

buffer

Description

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

kfree_skb

LINUX

Name

`kfree_skb` — free an sk_buff

Synopsis

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

Arguments

skb

buffer to free

Description

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

consume_skb

LINUX

Kernel Hackers ManualNovember 2026

Name

consume_skb — free an skbuff

Synopsis

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

Arguments

skb

buffer to free

Description

Drop a ref to the buffer and free it if the usage count has hit zero Functions identically to kfree_skb, but kfree_skb assumes that the frame is being dropped after a failure and notes that

skb_recycle_check

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_recycle_check` — check if skb can be reused for receive

Synopsis

```
bool skb_recycle_check (struct sk_buff * skb, int skb_size);
```

Arguments

skb

buffer

skb_size

minimum receive buffer size

Description

Checks that the skb passed in is not shared or cloned, and that it is linear and its head portion at least as large as skb_size so that it can be recycled as a receive buffer. If these conditions are met, this function does any necessary reference count dropping and cleans up the skbuff as if it just came from `__alloc_skb`.

skb_morph

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_morph` — morph one skb into another

Synopsis

```
struct sk_buff * skb_morph (struct sk_buff * dst, struct
sk_buff * src);
```

Arguments

`dst`

the skb to receive the contents

`src`

the skb to supply the contents

Description

This is identical to skb_clone except that the target skb is supplied by the user.

The target skb is returned upon exit.

skb_clone

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_clone` — duplicate an sk_buff

Synopsis

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

Arguments

`skb`

buffer to clone

`gfp_mask`

allocation priority

Description

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

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

skb_copy

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_copy` — create private copy of an sk_buff

Synopsis

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

Arguments

skb

buffer to copy

gfp_mask

allocation priority

Description

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

As by-product this function converts non-linear sk_buff to linear one, so that sk_buff becomes completely private and caller is allowed to modify all the data of

returned buffer. This means that this function is not recommended for use in circumstances when only header is going to be modified. Use `pskb_copy` instead.

pskb_copy

LINUX

Kernel Hackers ManualNovember 2026

Name

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

Synopsis

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

Arguments

skb

buffer to copy

gfp_mask

allocation priority

Description

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

pskb_expand_head

LINUX

Kernel Hackers ManualNovember 2026

Name

`pskb_expand_head` — reallocate header of sk_buff

Synopsis

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

Arguments

skb

buffer to reallocate

nhead

room to add at head

ntail

room to add at tail

gfp_mask

allocation priority

Description

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

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

skb_copy_expand

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_copy_expand` — copy and expand sk_buff

Synopsis

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

Arguments

skb

buffer to copy

newheadroom

new free bytes at head

newtailroom

new free bytes at tail

gfp_mask

allocation priority

Description

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

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

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

skb_pad

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_pad — zero pad the tail of an skb

Synopsis

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

Arguments

skb

buffer to pad

pad

space to pad

Description

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

May return error in out of memory cases. The skb is freed on error.

skb_put

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_put — add data to a buffer

Synopsis

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

Arguments

skb

buffer to use

len

amount of data to add

Description

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

skb_push

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_push — add data to the start of a buffer

Synopsis

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

Arguments

skb

buffer to use

len

amount of data to add

Description

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

skb_pull

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_pull — remove data from the start of a buffer

Synopsis

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

Arguments

skb

buffer to use

len

amount of data to remove

Description

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

skb_pull_data

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_pull_data` — remove data from the start of a buffer returning its original position.

Synopsis

```
void * skb_pull_data (struct sk_buff * skb, size_t len);
```

Arguments

skb

buffer to use

len

amount of data to remove

Description

This function removes data from the start of a buffer, returning the memory to the headroom. A pointer to the original data in the buffer is returned after checking if there is enough data to pull. Once the data has been pulled future pushes will overwrite the old data.

skb_trim

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_trim` — remove end from a buffer

Synopsis

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

Arguments

skb

buffer to alter

len

new length

Description

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

__pskb_pull_tail

LINUX

Kernel Hackers ManualNovember 2026

Name

__pskb_pull_tail — advance tail of skb header

Synopsis

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

Arguments

skb

buffer to reallocate

delta

number of bytes to advance tail

Description

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

sk_buff MUST have reference count of 1.

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

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

skb_store_bits

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_store_bits — store bits from kernel buffer to skb

Synopsis

```
int skb_store_bits (struct sk_buff * skb, int offset, const
void * from, int len);
```

Arguments

skb

destination buffer

offset

offset in destination

from

source buffer

len

number of bytes to copy

Description

Copy the specified number of bytes from the source buffer to the destination skb. This function handles all the messy bits of traversing fragment lists and such.

skb_dequeue

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_dequeue — remove from the head of the queue

Synopsis

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

Arguments

list

list to dequeue from

Description

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

skb_dequeue_tail

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_dequeue_tail — remove from the tail of the queue

Synopsis

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

Arguments

list

list to dequeue from

Description

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

skb_queue_purge

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_queue_purge` — empty a list

Synopsis

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

Arguments

`list`

list to empty

Description

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

skb_queue_head

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_queue_head` — queue a buffer at the list head

Synopsis

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

Arguments

`list`

list to use

`newsk`

buffer to queue

Description

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

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

skb_queue_tail

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_queue_tail` — queue a buffer at the list tail

Synopsis

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

Arguments

list

list to use

newsk

buffer to queue

Description

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

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

skb_unlink

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_unlink` — remove a buffer from a list

Synopsis

```
void skb_unlink (struct sk_buff * skb, struct sk_buff_head *
list);
```

Arguments

skb

buffer to remove

list

list to use

Description

Remove a packet from a list. The list locks are taken and this function is atomic with respect to other list locked calls

You must know what list the SKB is on.

skb_append

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_append — append a buffer

Synopsis

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

Arguments

old

buffer to insert after

newsk

buffer to insert

list

list to use

Description

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

skb_insert

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_insert` — insert a buffer

Synopsis

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

Arguments

`old`

buffer to insert before

`newsk`

buffer to insert

`list`

list to use

Description

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

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

skb_split

LINUX

Kernel Hackers ManualNovember 2026

Name

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

Synopsis

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

Arguments

skb

the buffer to split

skb1

the buffer to receive the second part

len

new length for skb

skb_prepare_seq_read

LINUX

Name

`skb_prepare_seq_read` — Prepare a sequential read of skb data

Synopsis

```
void skb_prepare_seq_read (struct sk_buff * skb, unsigned int
from, unsigned int to, struct skb_seq_state * st);
```

Arguments

skb

the buffer to read

from

lower offset of data to be read

to

upper offset of data to be read

st

state variable

Description

Initializes the specified state variable. Must be called before invoking `skb_seq_read` for the first time.

skb_seq_read

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_seq_read` — Sequentially read skb data

Synopsis

```
unsigned int skb_seq_read (unsigned int consumed, const u8 **
data, struct skb_seq_state * st);
```

Arguments

`consumed`

number of bytes consumed by the caller so far

`data`

destination pointer for data to be returned

`st`

state variable

Description

Reads a block of skb data at consumed relative to the lower offset specified to `skb_prepare_seq_read`. Assigns the head of the data block to data and returns the length of the block or 0 if the end of the skb data or the upper offset has been reached.

The caller is not required to consume all of the data returned, i.e. consumed is typically set to the number of bytes already consumed and the next call to `skb_seq_read` will return the remaining part of the block.

Note 1

The size of each block of data returned can be arbitrary, this limitation is the cost for zerocopy seqeuental reads of potentially non linear data.

Note 2

Fragment lists within fragments are not implemented at the moment, state->root_skb could be replaced with a stack for this purpose.

skb_abort_seq_read

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_abort_seq_read` — Abort a sequential read of skb data

Synopsis

```
void skb_abort_seq_read (struct skb_seq_state * st);
```

Arguments

st

state variable

Description

Must be called if `skb_seq_read` was not called until it returned 0.

skb_find_text

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_find_text` — Find a text pattern in skb data

Synopsis

```
unsigned int skb_find_text (struct sk_buff * skb, unsigned int
from, unsigned int to, struct ts_config * config, struct
ts_state * state);
```

Arguments

skb

the buffer to look in

from

search offset

to

search limit

config

textsearch configuration

state

uninitialized textsearch state variable

Description

Finds a pattern in the skb data according to the specified textsearch configuration. Use `textsearch_next` to retrieve subsequent occurrences of the pattern. Returns the offset to the first occurrence or UINT_MAX if no match was found.

skb_append_datato_frags

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_append_datato_frags` — append the user data to a skb

Synopsis

```
int skb_append_datato_frags (struct sock * sk, struct sk_buff
* skb, int (*getfrag) (void *from, char *to, int offset, int
len, int odd, struct sk_buff *skb), void * from, int length);
```

Arguments

sk

sock structure

skb

skb structure to be appened with user data.

getfrag

call back function to be used for getting the user data

from

pointer to user message iov

length

length of the iov message

Description

This procedure append the user data in the fragment part of the skb if any page alloc fails user this procedure returns -ENOMEM

skb_pull_rcsum

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_pull_rcsum` — pull skb and update receive checksum

Synopsis

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

Arguments

skb

buffer to update

len

length of data pulled

Description

This function performs an skb_pull on the packet and updates the CHECKSUM_COMPLETE checksum. It should be used on receive path processing instead of skb_pull unless you know that the checksum difference is zero (e.g., a valid IP header) or you are setting ip_summed to CHECKSUM_NONE.

skb_segment

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_segment` — Perform protocol segmentation on skb.

Synopsis

```
struct sk_buff * skb_segment (struct sk_buff * skb, u32
features);
```

Arguments

skb

buffer to segment

features

features for the output path (see dev->features)

Description

This function performs segmentation on the given skb. It returns a pointer to the first in a list of new skbs for the segments. In case of error it returns ERR_PTR(err).

skb_cow_data

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_cow_data` — Check that a socket buffer's data buffers are writable

Synopsis

```
int skb_cow_data (struct sk_buff * skb, int tailbits, struct
sk_buff ** trailer);
```

Arguments

`skb`

The socket buffer to check.

`tailbits`

Amount of trailing space to be added

`trailer`

Returned pointer to the skb where the *`tailbits`* space begins

Description

Make sure that the data buffers attached to a socket buffer are writable. If they are not, private copies are made of the data buffers and the socket buffer is set to use these instead.

If *tailbits* is given, make sure that there is space to write *tailbits* bytes of data beyond current end of socket buffer. *trailer* will be set to point to the skb in which this space begins.

The number of scatterlist elements required to completely map the COW'd and extended socket buffer will be returned.

skb_partial_csum_set

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_partial_csum_set — set up and verify partial csum values for packet

Synopsis

```
bool skb_partial_csum_set (struct sk_buff * skb, u16 start,
u16 off);
```

Arguments

skb

the skb to set

start

the number of bytes after skb->data to start checksumming.

off

the offset from start to place the checksum.

Description

For untrusted partially-checksummed packets, we need to make sure the values for skb->csum_start and skb->csum_offset are valid so we don't oops.

This function checks and sets those values and skb->ip_summed: if this returns false you should drop the packet.

sk_set_memalloc

LINUX

Kernel Hackers ManualNovember 2026

Name

`sk_set_memalloc` — sets `SOCK_MEMALLOC`

Synopsis

```
void sk_set_memalloc (struct sock * sk);
```

Arguments

sk

socket to set it on

Description

Set SOCK_MEMALLOC on a socket for access to emergency reserves. It's the responsibility of the admin to adjust min_free_kbytes to meet the requirements

sk_alloc

LINUX

Kernel Hackers ManualNovember 2026

Name

sk_alloc — All socket objects are allocated here

Synopsis

```
struct sock * sk_alloc (struct net * net, int family, gfp_t
priority, struct proto * prot);
```

Arguments

net

the applicable net namespace

family

protocol family

priority

for allocation (GFP_KERNEL, GFP_ATOMIC, etc)

prot

struct proto associated with this new sock instance

sk_wait_data

LINUX

Kernel Hackers ManualNovember 2026

Name

`sk_wait_data` — wait for data to arrive at sk_receive_queue

Synopsis

```
int sk_wait_data (struct sock * sk, long * timeo);
```

Arguments

sk

sock to wait on

timeo

for how long

Description

Now socket state including sk->sk_err is changed only under lock, hence we may omit checks after joining wait queue. We check receive queue before `schedule` only as optimization; it is very likely that `release_sock` added new data.

__sk_mem_schedule

LINUX

Kernel Hackers ManualNovember 2026

Name

__sk_mem_schedule — increase sk_forward_alloc and memory_allocated

Synopsis

```
int __sk_mem_schedule (struct sock * sk, int size, int kind);
```

Arguments

sk

socket

size

memory size to allocate

kind

allocation type

Description

If kind is SK_MEM_SEND, it means wmem allocation. Otherwise it means rmem allocation. This function assumes that protocols which have memory_pressure use sk_wmem_queued as write buffer accounting.

__sk_mem_reclaim

LINUX

Kernel Hackers ManualNovember 2026

Name

__sk_mem_reclaim — reclaim memory_allocated

Synopsis

```
void __sk_mem_reclaim (struct sock * sk);
```

Arguments

sk

socket

lock_sock_fast

LINUX

Kernel Hackers ManualNovember 2026

Name

lock_sock_fast — fast version of lock_sock

Synopsis

```
bool lock_sock_fast (struct sock * sk);
```

Arguments

sk

socket

Description

This version should be used for very small section, where process wont block return false if fast path is taken sk_lock.slock locked, owned = 0, BH disabled return true if slow path is taken sk_lock.slock unlocked, owned = 1, BH enabled

__skb_recv_datagram

LINUX

Kernel Hackers ManualNovember 2026

Name

__skb_recv_datagram — Receive a datagram skbuff

Synopsis

```
struct sk_buff * __skb_recv_datagram (struct sock * sk,
unsigned flags, int * peeked, int * err);
```

Arguments

sk

socket

flags

MSG_ flags

peeked

returns non-zero if this packet has been seen before

err

error code returned

Description

Get a datagram skbuff, understands the peeking, nonblocking wakeups and possible races. This replaces identical code in packet, raw and udp, as well as the IPX AX.25 and Appletalk. It also finally fixes the long standing peek and read race for datagram sockets. If you alter this routine remember it must be re-entrant.

This function will lock the socket if a skb is returned, so the caller needs to unlock the socket in that case (usually by calling skb_free_datagram)

* It does not lock socket since today. This function is * free of race conditions. This measure should/can improve * significantly datagram socket latencies at high loads, * when data copying to user space takes lots of time. * (BTW I've just killed the last `cli` in IP/IPv6/core/netlink/packet * 8) Great win.) * --ANK (980729)

The order of the tests when we find no data waiting are specified quite explicitly by POSIX 1003.1g, don't change them without having the standard around please.

skb_kill_datagram

LINUX

Name

`skb_kill_datagram` — Free a datagram skbuff forcibly

Synopsis

```
int skb_kill_datagram (struct sock * sk, struct sk_buff * skb,
unsigned int flags);
```

Arguments

sk

socket

skb

datagram skbuff

flags

MSG_ flags

Description

This function frees a datagram skbuff that was received by skb_recv_datagram. The flags argument must match the one used for skb_recv_datagram.

If the MSG_PEEK flag is set, and the packet is still on the receive queue of the socket, it will be taken off the queue before it is freed.

This function currently only disables BH when acquiring the sk_receive_queue lock. Therefore it must not be used in a context where that lock is acquired in an IRQ context.

It returns 0 if the packet was removed by us.

skb_copy_datagram_iovec

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_copy_datagram_iovec` — Copy a datagram to an iovec.

Synopsis

```
int skb_copy_datagram_iovec (const struct sk_buff * skb, int
offset, struct iovec * to, int len);
```

Arguments

skb

buffer to copy

offset

offset in the buffer to start copying from

to

io vector to copy to

len

amount of data to copy from buffer to iovec

Note

the iovec is modified during the copy.

skb_copy_datagram_const_iovec

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_copy_datagram_const_iovec` — Copy a datagram to an iovec.

Synopsis

```
int skb_copy_datagram_const_iovec (const struct sk_buff * skb,
int offset, const struct iovec * to, int to_offset, int len);
```

Arguments

skb

buffer to copy

offset

offset in the buffer to start copying from

to

io vector to copy to

to_offset

offset in the io vector to start copying to

len

amount of data to copy from buffer to iovec

Description

Returns 0 or -EFAULT.

Note

the iovec is not modified during the copy.

skb_copy_datagram_from_iovec

LINUX

Kernel Hackers ManualNovember 2026

Name

`skb_copy_datagram_from_iovec` — Copy a datagram from an iovec.

Synopsis

```
int skb_copy_datagram_from_iovec (struct sk_buff * skb, int
offset, const struct iovec * from, int from_offset, int len);
```

Arguments

skb

buffer to copy

offset

offset in the buffer to start copying to

from

io vector to copy to

from_offset

offset in the io vector to start copying from

len

amount of data to copy to buffer from iovec

Description

Returns 0 or -EFAULT.

Note

the iovec is not modified during the copy.

skb_copy_and_csum_datagram_iovec

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_copy_and_csum_datagram_iovec — Copy and checkum skb to user iovec.

Synopsis

```
int skb_copy_and_csum_datagram_iovec (struct sk_buff * skb,
int hlen, struct iovec * iov);
```

Arguments

skb

skbuff

hlen

hardware length

iov

io vector

Description

Caller _must_ check that skb will fit to this iovec.

Returns

0 - success. -EINVAL - checksum failure. -EFAULT - fault during copy. Beware, in this case iovec can be modified!

datagram_poll

LINUX

Kernel Hackers ManualNovember 2026

Name

`datagram_poll` — generic datagram poll

Synopsis

```
unsigned int datagram_poll (struct file * file, struct socket
* sock, poll_table * wait);
```

Arguments

file

file struct

sock

socket

wait

poll table

Datagram poll

Again totally generic. This also handles sequenced packet sockets providing the socket receive queue is only ever holding data ready to receive.

Note

when you _don't_ use this routine for this protocol, and you use a different write policy from `sock_writeable` then please supply your own write_space callback.

sk_stream_write_space

LINUX

Kernel Hackers ManualNovember 2026

Name

`sk_stream_write_space` — stream socket write_space callback.

Synopsis

```
void sk_stream_write_space (struct sock * sk);
```

Arguments

sk

socket

FIXME

write proper description

sk_stream_wait_connect

LINUX

Kernel Hackers ManualNovember 2026

Name

`sk_stream_wait_connect` — Wait for a socket to get into the connected state

Synopsis

```
int sk_stream_wait_connect (struct sock * sk, long * timeo_p);
```

Arguments

sk

sock to wait on

timeo_p

for how long to wait

Description

Must be called with the socket locked.

sk_stream_wait_memory

LINUX

Kernel Hackers ManualNovember 2026

Name

`sk_stream_wait_memory` — Wait for more memory for a socket

Synopsis

```
int sk_stream_wait_memory (struct sock * sk, long * timeo_p);
```

Arguments

sk

socket to wait for memory

timeo_p

for how long

1.3. Socket Filter

sk_filter

LINUX

Kernel Hackers ManualNovember 2026

Name

`sk_filter` — run a packet through a socket filter

Synopsis

```
int sk_filter (struct sock * sk, struct sk_buff * skb);
```

Arguments

sk

sock associated with sk_buff

skb

buffer to filter

Description

Run the filter code and then cut skb->data to correct size returned by sk_run_filter. If pkt_len is 0 we toss packet. If skb->len is smaller than pkt_len we keep whole skb->data. This is the socket level wrapper to sk_run_filter. It returns 0 if the packet should be accepted or -EPERM if the packet should be tossed.

sk_run_filter

LINUX

Kernel Hackers ManualNovember 2026

Name

`sk_run_filter` — run a filter on a socket

Synopsis

```
unsigned int sk_run_filter (const struct sk_buff * skb, const
struct sock_filter * fentry);
```

Arguments

skb

buffer to run the filter on

fentry

filter to apply

Description

Decode and apply filter instructions to the skb->data. Return length to keep, 0 for none. *skb* is the data we are filtering, *filter* is the array of filter instructions. Because all jumps are guaranteed to be before last instruction, and last instruction guaranteed to be a RET, we dont need to check flen. (We used to pass to this function the length of filter)

sk_chk_filter

LINUX

Kernel Hackers ManualNovember 2026

Name

`sk_chk_filter` — verify socket filter code

Synopsis

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

Arguments

`filter`

filter to verify

`flen`

length of filter

Description

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

All jumps are forward as they are not signed.

Returns 0 if the rule set is legal or -EINVAL if not.

sk_filter_release_rcu

LINUX

Kernel Hackers ManualNovember 2026

Name

`sk_filter_release_rcu` — Release a socket filter by rcu_head

Synopsis

```
void sk_filter_release_rcu (struct rcu_head * rcu);
```

Arguments

`rcu`

rcu_head that contains the sk_filter to free

sk_attach_filter

LINUX

Kernel Hackers ManualNovember 2026

Name

`sk_attach_filter` — attach a socket filter

Synopsis

```
int sk_attach_filter (struct sock_fprog * fprog, struct sock *
sk);
```

Arguments

fprog

the filter program

sk

the socket to use

Description

Attach the user's filter code. We first run some sanity checks on it to make sure it does not explode on us later. If an error occurs or there is insufficient memory for the filter a negative errno code is returned. On success the return is zero.

1.4. Generic Network Statistics

struct gnet_stats_basic

LINUX

Kernel Hackers ManualNovember 2026

Name

`struct gnet_stats_basic` — byte/packet throughput statistics

Synopsis

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

Members

bytes

number of seen bytes

packets

number of seen packets

struct gnet_stats_rate_est

LINUX

Kernel Hackers ManualNovember 2026

Name

`struct gnet_stats_rate_est` — rate estimator

Synopsis

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

Members

bps

current byte rate

pps

current packet rate

struct gnet_stats_queue

LINUX

Kernel Hackers ManualNovember 2026

Name

`struct gnet_stats_queue` — queuing statistics

Synopsis

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

Members

qlen

queue length

backlog

backlog size of queue

drops

number of dropped packets

requeues

number of requeues

overlimits

number of enqueues over the limit

struct gnet_estimator

LINUX

Kernel Hackers ManualNovember 2026

Name

`struct gnet_estimator` — rate estimator configuration

Synopsis

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

Members

interval

sampling period

ewma_log

the log of measurement window weight

gnet_stats_start_copy_compat

LINUX

Kernel Hackers ManualNovember 2026

Name

`gnet_stats_start_copy_compat` — start dumping procedure in compatibility mode

Synopsis

```
int gnet_stats_start_copy_compat (struct sk_buff * skb, int
type, int tc_stats_type, int xstats_type, spinlock_t * lock,
struct gnet_dump * d);
```

Arguments

skb

socket buffer to put statistics TLVs into

type

TLV type for top level statistic TLV

tc_stats_type

TLV type for backward compatibility struct tc_stats TLV

xstats_type

TLV type for backward compatibility xstats TLV

lock

statistics lock

d

dumping handle

Description

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

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

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

gnet_stats_start_copy

LINUX

Kernel Hackers ManualNovember 2026

Name

`gnet_stats_start_copy` — start dumping procedure in compatibility mode

Synopsis

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

Arguments

skb

socket buffer to put statistics TLVs into

type

TLV type for top level statistic TLV

lock

statistics lock

d

dumping handle

Description

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

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

gnet_stats_copy_basic

LINUX

Kernel Hackers ManualNovember 2026

Name

`gnet_stats_copy_basic` — copy basic statistics into statistic TLV

Synopsis

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

Arguments

d

dumping handle

b

basic statistics

Description

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

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

gnet_stats_copy_rate_est

LINUX

Kernel Hackers ManualNovember 2026

Name

`gnet_stats_copy_rate_est` — copy rate estimator statistics into statistics TLV

Synopsis

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

Arguments

d

dumping handle

b

basic statistics

r

rate estimator statistics

Description

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

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

gnet_stats_copy_queue

LINUX

Kernel Hackers ManualNovember 2026

Name

`gnet_stats_copy_queue` — copy queue statistics into statistics TLV

Synopsis

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

Arguments

d

dumping handle

q

queue statistics

Description

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

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

gnet_stats_copy_app

LINUX

Kernel Hackers ManualNovember 2026

Name

`gnet_stats_copy_app` — copy application specific statistics into statistics TLV

Synopsis

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

Arguments

d

dumping handle

st

application specific statistics data

len

length of data

Description

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

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

gnet_stats_finish_copy

LINUX

Kernel Hackers ManualNovember 2026

Name

`gnet_stats_finish_copy` — finish dumping procedure

Synopsis

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

Arguments

d

dumping handle

Description

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

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

gen_new_estimator

LINUX

Kernel Hackers ManualNovember 2026

Name

`gen_new_estimator` — create a new rate estimator

Synopsis

```
int gen_new_estimator (struct gnet_stats_basic_packed *
bstats, struct gnet_stats_rate_est * rate_est, spinlock_t *
stats_lock, struct nlattr * opt);
```

Arguments

`bstats`

basic statistics

`rate_est`

rate estimator statistics

`stats_lock`

statistics lock

`opt`

rate estimator configuration TLV

Description

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

Returns 0 on success or a negative error code.

gen_kill_estimator

LINUX

Kernel Hackers ManualNovember 2026

Name

`gen_kill_estimator` — remove a rate estimator

Synopsis

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

Arguments

`bstats`

basic statistics

`rate_est`

rate estimator statistics

Description

Removes the rate estimator specified by bstats and rate_est.

Note

Caller should respect an RCU grace period before freeing stats_lock

gen_replace_estimator

LINUX

Kernel Hackers ManualNovember 2026

Name

`gen_replace_estimator` — replace rate estimator configuration

Synopsis

```
int gen_replace_estimator (struct gnet_stats_basic_packed *
bstats, struct gnet_stats_rate_est * rate_est, spinlock_t *
stats_lock, struct nlattr * opt);
```

Arguments

`bstats`

basic statistics

`rate_est`

rate estimator statistics

stats_lock

statistics lock

opt

rate estimator configuration TLV

Description

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

Returns 0 on success or a negative error code.

gen_estimator_active

LINUX

Kernel Hackers ManualNovember 2026

Name

`gen_estimator_active` — test if estimator is currently in use

Synopsis

```
bool gen_estimator_active (const struct
gnet_stats_basic_packed * bstats, const struct
gnet_stats_rate_est * rate_est);
```

Arguments

bstats

basic statistics

rate_est

rate estimator statistics

Description

Returns true if estimator is active, and false if not.

1.5. SUN RPC subsystem

xdr_encode_opaque_fixed

LINUX

Kernel Hackers ManualNovember 2026

Name

`xdr_encode_opaque_fixed` — Encode fixed length opaque data

Synopsis

```
__be32 * xdr_encode_opaque_fixed (__be32 * p, const void *
ptr, unsigned int nbytes);
```

Arguments

p

pointer to current position in XDR buffer.

ptr

pointer to data to encode (or NULL)

nbytes

size of data.

Description

Copy the array of data of length nbytes at ptr to the XDR buffer at position p, then align to the next 32-bit boundary by padding with zero bytes (see RFC1832).

Note

if ptr is NULL, only the padding is performed.

Returns the updated current XDR buffer position

xdr_encode_opaque

LINUX

Kernel Hackers ManualNovember 2026

Name

`xdr_encode_opaque` — Encode variable length opaque data

Synopsis

```
__be32 * xdr_encode_opaque (__be32 * p, const void * ptr,
unsigned int nbytes);
```

Arguments

p

pointer to current position in XDR buffer.

ptr

pointer to data to encode (or NULL)

nbytes

size of data.

Description

Returns the updated current XDR buffer position

xdr_terminate_string

LINUX

Kernel Hackers ManualNovember 2026

Name

`xdr_terminate_string` — '\0'-terminate a string residing in an xdr_buf

Synopsis

```
void xdr_terminate_string (struct xdr_buf * buf, const u32
len);
```

Arguments

buf

XDR buffer where string resides

len

length of string, in bytes

xdr_init_encode

LINUX

Kernel Hackers ManualNovember 2026

Name

`xdr_init_encode` — Initialize a struct xdr_stream for sending data.

Synopsis

```
void xdr_init_encode (struct xdr_stream * xdr, struct xdr_buf
* buf, __be32 * p);
```

Arguments

xdr

pointer to xdr_stream struct

buf

pointer to XDR buffer in which to encode data

p

current pointer inside XDR buffer

Note

at the moment the RPC client only passes the length of our scratch buffer in the xdr_buf's header kvec. Previously this meant we needed to call `xdr_adjust_iovec` after encoding the data. With the new scheme, the xdr_stream manages the details of the buffer length, and takes care of adjusting the kvec length for us.

xdr_reserve_space

LINUX

Kernel Hackers ManualNovember 2026

Name

`xdr_reserve_space` — Reserve buffer space for sending

Synopsis

```
__be32 * xdr_reserve_space (struct xdr_stream * xdr, size_t
nbytes);
```

Arguments

xdr

pointer to xdr_stream

nbytes

number of bytes to reserve

Description

Checks that we have enough buffer space to encode 'nbytes' more bytes of data. If so, update the total xdr_buf length, and adjust the length of the current kvec.

xdr_write_pages

LINUX

Kernel Hackers ManualNovember 2026

Name

`xdr_write_pages` — Insert a list of pages into an XDR buffer for sending

Synopsis

```
void xdr_write_pages (struct xdr_stream * xdr, struct page ** pages, unsigned int base, unsigned int len);
```

Arguments

xdr

pointer to xdr_stream

pages

list of pages

base

offset of first byte

len

length of data in bytes

xdr_init_decode

LINUX

Kernel Hackers ManualNovember 2026

Name

xdr_init_decode — Initialize an xdr_stream for decoding data.

Synopsis

```
void xdr_init_decode (struct xdr_stream * xdr, struct xdr_buf
* buf, __be32 * p);
```

Arguments

xdr

pointer to xdr_stream struct

buf

pointer to XDR buffer from which to decode data

p

current pointer inside XDR buffer

xdr_init_decode_pages

LINUX

Kernel Hackers ManualNovember 2026

Name

xdr_init_decode_pages — Initialize an xdr_stream for decoding data.

Synopsis

```
void xdr_init_decode_pages (struct xdr_stream * xdr, struct
xdr_buf * buf, struct page ** pages, unsigned int len);
```

Arguments

xdr

pointer to xdr_stream struct

buf

pointer to XDR buffer from which to decode data

pages

list of pages to decode into

len

length in bytes of buffer in pages

xdr_set_scratch_buffer

LINUX

Name

`xdr_set_scratch_buffer` — Attach a scratch buffer for decoding data.

Synopsis

```
void xdr_set_scratch_buffer (struct xdr_stream * xdr, void *
buf, size_t buflen);
```

Arguments

`xdr`

pointer to xdr_stream struct

`buf`

pointer to an empty buffer

`buflen`

size of ’buf’

Description

The scratch buffer is used when decoding from an array of pages. If an `xdr_inline_decode` call spans across page boundaries, then we copy the data into the scratch buffer in order to allow linear access.

xdr_inline_decode

LINUX

Name

`xdr_inline_decode` — Retrieve XDR data to decode

Synopsis

```
__be32 * xdr_inline_decode (struct xdr_stream * xdr, size_t
nbytes);
```

Arguments

xdr

pointer to xdr_stream struct

nbytes

number of bytes of data to decode

Description

Check if the input buffer is long enough to enable us to decode 'nbytes' more bytes of data starting at the current position. If so return the current pointer, then update the current pointer position.

xdr_read_pages

LINUX

Name

`xdr_read_pages` — Ensure page-based XDR data to decode is aligned at current pointer position

Synopsis

```
void xdr_read_pages (struct xdr_stream * xdr, unsigned int len);
```

Arguments

`xdr`

pointer to xdr_stream struct

`len`

number of bytes of page data

Description

Moves data beyond the current pointer position from the XDR head[] buffer into the page list. Any data that lies beyond current position + “len” bytes is moved into the XDR tail[].

xdr_enter_page

LINUX

Name

`xdr_enter_page` — decode data from the XDR page

Synopsis

```
void xdr_enter_page (struct xdr_stream * xdr, unsigned int
len);
```

Arguments

xdr

pointer to xdr_stream struct

len

number of bytes of page data

Description

Moves data beyond the current pointer position from the XDR head[] buffer into the page list. Any data that lies beyond current position + “len” bytes is moved into the XDR tail[]. The current pointer is then repositioned at the beginning of the first XDR page.

svc_print_addr

LINUX

Name

`svc_print_addr` — Format rq_addr field for printing

Synopsis

```
char * svc_print_addr (struct svc_rqst * rqstp, char * buf,
size_t len);
```

Arguments

`rqstp`

svc_rqst struct containing address to print

`buf`

target buffer for formatted address

`len`

length of target buffer

svc_reserve

LINUX

Name

`svc_reserve` — change the space reserved for the reply to a request.

Synopsis

```
void svc_reserve (struct svc_rqst * rqstp, int space);
```

Arguments

rqstp

The request in question

space

new max space to reserve

Description

Each request reserves some space on the output queue of the transport to make sure the reply fits. This function reduces that reserved space to be the amount of space used already, plus *space*.

svc_find_xprt

LINUX

Kernel Hackers ManualNovember 2026

Name

svc_find_xprt — find an RPC transport instance

Synopsis

```
struct svc_xprt * svc_find_xprt (struct svc_serv * serv, const
char * xcl_name, const sa_family_t af, const unsigned short
```

```
port);
```

Arguments

serv

pointer to svc_serv to search

xcl_name

C string containing transport's class name

af

Address family of transport's local address

port

transport's IP port number

Description

Return the transport instance pointer for the endpoint accepting connections/peer traffic from the specified transport class, address family and port.

Specifying 0 for the address family or port is effectively a wild-card, and will result in matching the first transport in the service's list that has a matching class name.

svc_xprt_names

LINUX

Kernel Hackers ManualNovember 2026

Name

`svc_xprt_names` — format a buffer with a list of transport names

Synopsis

```
int svc_xprt_names (struct svc_serv * serv, char * buf, const
int buflen);
```

Arguments

serv

pointer to an RPC service

buf

pointer to a buffer to be filled in

buflen

length of buffer to be filled in

Description

Fills in *buf* with a string containing a list of transport names, each name terminated with '\n'.

Returns positive length of the filled-in string on success; otherwise a negative errno value is returned if an error occurs.

xprt_register_transport

LINUX

Kernel Hackers ManualNovember 2026

Name

xprt_register_transport — register a transport implementation

Synopsis

```
int xprt_register_transport (struct xprt_class * transport);
```

Arguments

transport

transport to register

Description

If a transport implementation is loaded as a kernel module, it can call this interface to make itself known to the RPC client.

0

transport successfully registered -EEXIST: transport already registered -EINVAL: transport module being unloaded

xprt_unregister_transport

LINUX

Kernel Hackers ManualNovember 2026

Name

`xprt_unregister_transport` — unregister a transport implementation

Synopsis

```
int xprt_unregister_transport (struct xprt_class * transport);
```

Arguments

transport

transport to unregister

0

transport successfully unregistered -ENOENT: transport never registered

xprt_load_transport

LINUX

Kernel Hackers ManualNovember 2026

Name

xprt_load_transport — load a transport implementation

Synopsis

```
int xprt_load_transport (const char * transport_name);
```

Arguments

transport_name

transport to load

0

transport successfully loaded -ENOENT: transport module not available

xprt_release_xprt

LINUX

Kernel Hackers ManualNovember 2026

Name

xprt_release_xprt — allow other requests to use a transport

Synopsis

```
void xprt_release_xprt (struct rpc_xprt * xprt, struct
rpc_task * task);
```

Arguments

xprt

transport with other tasks potentially waiting

task

task that is releasing access to the transport

Description

Note that “task” can be NULL. No congestion control is provided.

xprt_release_xprt_cong

LINUX

Kernel Hackers ManualNovember 2026

Name

`xprt_release_xprt_cong` — allow other requests to use a transport

Synopsis

```
void xprt_release_xprt_cong (struct rpc_xprt * xprt, struct
rpc_task * task);
```

Arguments

xprt

transport with other tasks potentially waiting

task

task that is releasing access to the transport

Description

Note that “task” can be NULL. Another task is awoken to use the transport if the transport’s congestion window allows it.

xprt_release_rqst_cong

LINUX

Kernel Hackers ManualNovember 2026

Name

xprt_release_rqst_cong — housekeeping when request is complete

Synopsis

```
void xprt_release_rqst_cong (struct rpc_task * task);
```

Arguments

task

RPC request that recently completed

Description

Useful for transports that require congestion control.

xprt_adjust_cwnd

LINUX

Kernel Hackers ManualNovember 2026

Name

xprt_adjust_cwnd — adjust transport congestion window

Synopsis

```
void xprt_adjust_cwnd (struct rpc_task * task, int result);
```

Arguments

task

recently completed RPC request used to adjust window

result

result code of completed RPC request

Description

We use a time-smoothed congestion estimator to avoid heavy oscillation.

xprt_wake_pending_tasks

LINUX

Kernel Hackers ManualNovember 2026

Name

xprt_wake_pending_tasks — wake all tasks on a transport's pending queue

Synopsis

```
void xprt_wake_pending_tasks (struct rpc_xprt * xprt, int
status);
```

Arguments

xprt

transport with waiting tasks

status

result code to plant in each task before waking it

xprt_wait_for_buffer_space

LINUX

Kernel Hackers ManualNovember 2026

Name

xprt_wait_for_buffer_space — wait for transport output buffer to clear

Synopsis

```
void xprt_wait_for_buffer_space (struct rpc_task * task,
rpc_action action);
```

Arguments

task

task to be put to sleep

action

function pointer to be executed after wait

Description

Note that we only set the timer for the case of `RPC_IS_SOFT`, since we don't in general want to force a socket disconnection due to an incomplete RPC call transmission.

xprt_write_space

LINUX

Kernel Hackers ManualNovember 2026

Name

`xprt_write_space` — wake the task waiting for transport output buffer space

Synopsis

```
void xprt_write_space (struct rpc_xprt * xprt);
```

Arguments

`xprt`

transport with waiting tasks

Description

Can be called in a soft IRQ context, so xprt_write_space never sleeps.

xprt_set_retrans_timeout_def

LINUX

Kernel Hackers ManualNovember 2026

Name

`xprt_set_retrans_timeout_def` — set a request's retransmit timeout

Synopsis

```
void xprt_set_retrans_timeout_def (struct rpc_task * task);
```

Arguments

`task`

task whose timeout is to be set

Description

Set a request's retransmit timeout based on the transport's default timeout parameters. Used by transports that don't adjust the retransmit timeout based on round-trip time estimation.

xprt_disconnect_done

LINUX

Name

xprt_disconnect_done — mark a transport as disconnected

Synopsis

```
void xprt_disconnect_done (struct rpc_xprt * xprt);
```

Arguments

xprt

transport to flag for disconnect

xprt_lookup_rqst

LINUX

Name

xprt_lookup_rqst — find an RPC request corresponding to an XID

Synopsis

```
struct rpc_rqst * xprt_lookup_rqst (struct rpc_xprt * xprt,
__be32 xid);
```

Arguments

xprt

transport on which the original request was transmitted

xid

RPC XID of incoming reply

xprt_complete_rqst

LINUX

Kernel Hackers ManualNovember 2026

Name

xprt_complete_rqst — called when reply processing is complete

Synopsis

```
void xprt_complete_rqst (struct rpc_task * task, int copied);
```

Arguments

task

RPC request that recently completed

copied

actual number of bytes received from the transport

Description

Caller holds transport lock.

rpc_wake_up

LINUX

Kernel Hackers ManualNovember 2026

Name

`rpc_wake_up` — wake up all rpc_tasks

Synopsis

```
void rpc_wake_up (struct rpc_wait_queue * queue);
```

Arguments

`queue`

rpc_wait_queue on which the tasks are sleeping

Description

Grabs queue->lock

rpc_wake_up_status

LINUX

Kernel Hackers ManualNovember 2026

Name

`rpc_wake_up_status` — wake up all rpc_tasks and set their status value.

Synopsis

```
void rpc_wake_up_status (struct rpc_wait_queue * queue, int
status);
```

Arguments

`queue`

rpc_wait_queue on which the tasks are sleeping

`status`

status value to set

Description

Grabs queue->lock

rpc_malloc

LINUX

Name

`rpc_malloc` — allocate an RPC buffer

Synopsis

```
void * rpc_malloc (struct rpc_task * task, size_t size);
```

Arguments

`task`

RPC task that will use this buffer

`size`

requested byte size

Description

To prevent rpciod from hanging, this allocator never sleeps, returning NULL and suppressing warning if the request cannot be serviced immediately. The caller can arrange to sleep in a way that is safe for rpciod.

Most requests are 'small' (under 2KiB) and can be serviced from a mempool, ensuring that NFS reads and writes can always proceed, and that there is good locality of reference for these buffers.

In order to avoid memory starvation triggering more writebacks of NFS requests, we avoid using GFP_KERNEL.

rpc_free

LINUX

Kernel Hackers ManualNovember 2026

Name

`rpc_free` — free buffer allocated via rpc_malloc

Synopsis

```
void rpc_free (void * buffer);
```

Arguments

`buffer`

buffer to free

xdr_skb_read_bits

LINUX

Kernel Hackers ManualNovember 2026

Name

`xdr_skb_read_bits` — copy some data bits from skb to internal buffer

Synopsis

```
size_t xdr_skb_read_bits (struct xdr_skb_reader * desc, void *
to, size_t len);
```

Arguments

desc

sk_buff copy helper

to

copy destination

len

number of bytes to copy

Description

Possibly called several times to iterate over an sk_buff and copy data out of it.

xdr_partial_copy_from_skb

LINUX

Kernel Hackers ManualNovember 2026

Name

`xdr_partial_copy_from_skb` — copy data out of an skb

Synopsis

```
ssize_t xdr_partial_copy_from_skb (struct xdr_buf * xdr,
unsigned int base, struct xdr_skb_reader * desc,
xdr_skb_read_actor copy_actor);
```

Arguments

xdr

target XDR buffer

base

starting offset

desc

sk_buff copy helper

copy_actor

virtual method for copying data

csum_partial_copy_to_xdr

LINUX

Kernel Hackers ManualNovember 2026

Name

csum_partial_copy_to_xdr — checksum and copy data

Synopsis

```
int csum_partial_copy_to_xdr (struct xdr_buf * xdr, struct
sk_buff * skb);
```

Arguments

xdr

target XDR buffer

skb

source skb

Description

We have set things up such that we perform the checksum of the UDP packet in parallel with the copies into the RPC client iovec. -DaveM

rpc_alloc_iostats

LINUX

Kernel Hackers ManualNovember 2026

Name

`rpc_alloc_iostats` — allocate an rpc_iostats structure

Synopsis

```
struct rpc_iostats * rpc_alloc_iostats (struct rpc_clnt *
clnt);
```

Arguments

clnt

RPC program, version, and xprt

rpc_free_iostats

LINUX

Kernel Hackers ManualNovember 2026

Name

rpc_free_iostats — release an rpc_iostats structure

Synopsis

```
void rpc_free_iostats (struct rpc_iostats * stats);
```

Arguments

stats

doomed rpc_iostats structure

rpc_queue_upcall

LINUX

Kernel Hackers ManualNovember 2026

Name

`rpc_queue_upcall` — queue an upcall message to userspace

Synopsis

```
int rpc_queue_upcall (struct inode * inode, struct
rpc_pipe_msg * msg);
```

Arguments

`inode`

inode of upcall pipe on which to queue given message

`msg`

message to queue

Description

Call with an *`inode`* created by `rpc_mkpipe` to queue an upcall. A userspace process may then later read the upcall by performing a read on an open file for this inode. It is up to the caller to initialize the fields of *`msg`* (other than *`msg`*->list) appropriately.

rpc_mkpipe

LINUX

Kernel Hackers ManualNovember 2026

Name

`rpc_mkpipe` — make an rpc_pipefs file for kernel<->userspace communication

Synopsis

```
struct dentry * rpc_mkpipe (struct dentry * parent, const char
* name, void * private, const struct rpc_pipe_ops * ops, int
flags);
```

Arguments

parent

dentry of directory to create new “pipe” in

name

name of pipe

private

private data to associate with the pipe, for the caller’s use

ops

operations defining the behavior of the pipe: upcall, downcall, release_pipe, open_pipe, and destroy_msg.

flags

rpc_inode flags

Description

Data is made available for userspace to read by calls to `rpc_queue_upcall`. The actual reads will result in calls to *`ops`*->upcall, which will be called with the file pointer, message, and userspace buffer to copy to.

Writes can come at any time, and do not necessarily have to be responses to upcalls. They will result in calls to *`msg`*->downcall.

The *`private`* argument passed here will be available to all these methods from the file pointer, via RPC_I(file->f_dentry->d_inode)->private.

rpc_unlink

LINUX

Kernel Hackers ManualNovember 2026

Name

`rpc_unlink` — remove a pipe

Synopsis

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

Arguments

`dentry`

dentry for the pipe, as returned from rpc_mkpipe

Description

After this call, lookups will no longer find the pipe, and any attempts to read or write using preexisting opens of the pipe will return -EPIPE.

rpcb_getport_async

LINUX

Kernel Hackers ManualNovember 2026

Name

`rpcb_getport_async` — obtain the port for a given RPC service on a given host

Synopsis

```
void rpcb_getport_async (struct rpc_task * task);
```

Arguments

`task`

task that is waiting for portmapper request

Description

This one can be called for an ongoing RPC request, and can be used in an async (rpciod) context.

rpc_bind_new_program

LINUX

Kernel Hackers ManualNovember 2026

Name

rpc_bind_new_program — bind a new RPC program to an existing client

Synopsis

```
struct rpc_clnt * rpc_bind_new_program (struct rpc_clnt * old,
struct rpc_program * program, u32 vers);
```

Arguments

old

old rpc_client

program

rpc program to set

vers

rpc program version

Description

Clones the rpc client and sets up a new RPC program. This is mainly of use for enabling different RPC programs to share the same transport. The Sun NFSv2/v3 ACL protocol can do this.

rpc_run_task

LINUX

Kernel Hackers ManualNovember 2026

Name

`rpc_run_task` — Allocate a new RPC task, then run rpc_execute against it

Synopsis

```
struct rpc_task * rpc_run_task (const struct rpc_task_setup *
task_setup_data);
```

Arguments

`task_setup_data`

pointer to task initialisation data

rpc_call_sync

LINUX

Kernel Hackers ManualNovember 2026

Name

`rpc_call_sync` — Perform a synchronous RPC call

Synopsis

```
int rpc_call_sync (struct rpc_clnt * clnt, const struct
rpc_message * msg, int flags);
```

Arguments

clnt

pointer to RPC client

msg

RPC call parameters

flags

RPC call flags

rpc_call_async

LINUX

Kernel Hackers ManualNovember 2026

Name

rpc_call_async — Perform an asynchronous RPC call

Synopsis

```
int rpc_call_async (struct rpc_clnt * clnt, const struct
rpc_message * msg, int flags, const struct rpc_call_ops *
tk_ops, void * data);
```

Arguments

clnt

pointer to RPC client

msg

RPC call parameters

flags

RPC call flags

tk_ops

RPC call ops

data

user call data

rpc_peeraddr

LINUX

Kernel Hackers ManualNovember 2026

Name

rpc_peeraddr — extract remote peer address from clnt's xprt

Synopsis

```
size_t rpc_peeraddr (struct rpc_clnt * clnt, struct sockaddr *
buf, size_t bufsize);
```

Arguments

clnt

RPC client structure

buf

target buffer

bufsize

length of target buffer

Description

Returns the number of bytes that are actually in the stored address.

rpc_peeraddr2str

LINUX

Kernel Hackers ManualNovember 2026

Name

`rpc_peeraddr2str` — return remote peer address in printable format

Synopsis

```
const char * rpc_peeraddr2str (struct rpc_clnt * clnt, enum
rpc_display_format_t format);
```

Arguments

clnt

RPC client structure

format

address format

rpc_force_rebind

LINUX

Kernel Hackers ManualNovember 2026

Name

rpc_force_rebind — force transport to check that remote port is unchanged

Synopsis

```
void rpc_force_rebind (struct rpc_clnt * clnt);
```

Arguments

clnt

client to rebind

1.6. WiMAX

wimax_msg_alloc

LINUX

Kernel Hackers ManualNovember 2026

Name

`wimax_msg_alloc` — Create a new skb for sending a message to userspace

Synopsis

```
struct sk_buff * wimax_msg_alloc (struct wimax_dev *
wimax_dev, const char * pipe_name, const void * msg, size_t
size, gfp_t gfp_flags);
```

Arguments

wimax_dev

WiMAX device descriptor

pipe_name

"named pipe" the message will be sent to

msg

pointer to the message data to send

size

size of the message to send (in bytes), including the header.

gfp_flags

flags for memory allocation.

Returns

`0` if ok, negative errno code on error

Description

Allocates an skb that will contain the message to send to user space over the messaging pipe and initializes it, copying the payload.

Once this call is done, you can deliver it with `wimax_msg_send`.

IMPORTANT

Don't use `skb_push`/`skb_pull`/`skb_reserve` on the skb, as `wimax_msg_send` depends on skb->data being placed at the beginning of the user message.

Unlike other WiMAX stack calls, this call can be used way early, even before `wimax_dev_add` is called, as long as the wimax_dev->net_dev pointer is set to point to a proper net_dev. This is so that drivers can use it early in case they need to send stuff around or communicate with user space.

wimax_msg_data_len

LINUX

Kernel Hackers ManualNovember 2026

Name

`wimax_msg_data_len` — Return a pointer and size of a message's payload

Synopsis

```
const void * wimax_msg_data_len (struct sk_buff * msg, size_t
* size);
```

Arguments

msg

Pointer to a message created with `wimax_msg_alloc`

size

Pointer to where to store the message's size

Description

Returns the pointer to the message data.

wimax_msg_data

LINUX

Kernel Hackers ManualNovember 2026

Name

`wimax_msg_data` — Return a pointer to a message's payload

Synopsis

```
const void * wimax_msg_data (struct sk_buff * msg);
```

Arguments

msg

Pointer to a message created with `wimax_msg_alloc`

wimax_msg_len

LINUX

Kernel Hackers ManualNovember 2026

Name

`wimax_msg_len` — Return a message's payload length

Synopsis

```
ssize_t wimax_msg_len (struct sk_buff * msg);
```

Arguments

msg

Pointer to a message created with `wimax_msg_alloc`

wimax_msg_send

LINUX

Name

`wimax_msg_send` — Send a pre-allocated message to user space

Synopsis

```
int wimax_msg_send (struct wimax_dev * wimax_dev, struct
sk_buff * skb);
```

Arguments

`wimax_dev`

WiMAX device descriptor

`skb`

struct sk_buff returned by `wimax_msg_alloc`. Note the ownership of *`skb`* is transferred to this function.

Returns

0 if ok, < 0 errno code on error

Description

Sends a free-form message that was preallocated with `wimax_msg_alloc` and filled up.

Assumes that once you pass an skb to this function for sending, it owns it and will release it when done (on success).

IMPORTANT

Don't use `skb_push`/`skb_pull`/`skb_reserve` on the skb, as `wimax_msg_send` depends on skb->data being placed at the beginning of the user message.

Unlike other WiMAX stack calls, this call can be used way early, even before `wimax_dev_add` is called, as long as the wimax_dev->net_dev pointer is set to point to a proper net_dev. This is so that drivers can use it early in case they need to send stuff around or communicate with user space.

wimax_msg

LINUX

Kernel Hackers ManualNovember 2026

Name

`wimax_msg` — Send a message to user space

Synopsis

```
int wimax_msg (struct wimax_dev * wimax_dev, const char *
pipe_name, const void * buf, size_t size, gfp_t gfp_flags);
```

Arguments

`wimax_dev`

WiMAX device descriptor (properly referenced)

`pipe_name`

"named pipe" the message will be sent to

`buf`

pointer to the message to send.

size

size of the buffer pointed to by *buf* (in bytes).

gfp_flags

flags for memory allocation.

Returns

0 if ok, negative errno code on error.

Description

Sends a free-form message to user space on the device *wimax_dev*.

NOTES

Once the *skb* is given to this function, who will own it and will release it when done (unless it returns error).

wimax_reset

LINUX

Kernel Hackers ManualNovember 2026

Name

wimax_reset — Reset a WiMAX device

Synopsis

```
int wimax_reset (struct wimax_dev * wimax_dev);
```

Arguments

wimax_dev

WiMAX device descriptor

Returns

0 if ok and a warm reset was done (the device still exists in the system).

-ENODEV if a cold/bus reset had to be done (device has disconnected and reconnected, so current handle is not valid any more).

-EINVAL if the device is not even registered.

Any other negative error code shall be considered as non-recoverable.

Description

Called when wanting to reset the device for any reason. Device is taken back to power on status.

This call blocks; on successful return, the device has completed the reset process and is ready to operate.

wimax_report_rfkill_hw

LINUX

Kernel Hackers ManualNovember 2026

Name

wimax_report_rfkill_hw — Reports changes in the hardware RF switch

Synopsis

```
void wimax_report_rfkill_hw (struct wimax_dev * wimax_dev,
enum wimax_rf_state state);
```

Arguments

wimax_dev

WiMAX device descriptor

state

New state of the RF Kill switch. WIMAX_RF_ON radio on, WIMAX_RF_OFF radio off.

Description

When the device detects a change in the state of thehardware RF switch, it must call this function to let the WiMAX kernel stack know that the state has changed so it can be properly propagated.

The WiMAX stack caches the state (the driver doesn't need to). As well, as the change is propagated it will come back as a request to change the software state to mirror the hardware state.

If the device doesn't have a hardware kill switch, just report it on initialization as always on (WIMAX_RF_ON, radio on).

wimax_report_rfkill_sw

LINUX

Name

`wimax_report_rfkill_sw` — Reports changes in the software RF switch

Synopsis

```
void wimax_report_rfkill_sw (struct wimax_dev * wimax_dev,
enum wimax_rf_state state);
```

Arguments

`wimax_dev`

WiMAX device descriptor

`state`

New state of the RF kill switch. `WIMAX_RF_ON` radio on, `WIMAX_RF_OFF` radio off.

Description

Reports changes in the software RF switch state to the the WiMAX stack.

The main use is during initialization, so the driver can query the device for its current software radio kill switch state and feed it to the system.

On the side, the device does not change the software state by itself. In practice, this can happen, as the device might decide to switch (in software) the radio off for different reasons.

wimax_rfkill

LINUX

Kernel Hackers ManualNovember 2026

Name

`wimax_rfkill` — Set the software RF switch state for a WiMAX device

Synopsis

```
int wimax_rfkill (struct wimax_dev * wimax_dev, enum
wimax_rf_state state);
```

Arguments

`wimax_dev`

WiMAX device descriptor

`state`

New RF state.

Returns

>= 0 toggle state if ok, < 0 errno code on error. The toggle state is returned as a bitmap, bit 0 being the hardware RF state, bit 1 the software RF state.

0 means disabled (`WIMAX_RF_ON`, radio on), 1 means enabled radio off (`WIMAX_RF_OFF`).

Description

Called by the user when he wants to request the WiMAX radio to be switched on (`WIMAX_RF_ON`) or off (`WIMAX_RF_OFF`). With `WIMAX_RF_QUERY`, just the current state is returned.

NOTE

This call will block until the operation is complete.

wimax_state_change

LINUX

Kernel Hackers ManualNovember 2026

Name

`wimax_state_change` — Set the current state of a WiMAX device

Synopsis

```
void wimax_state_change (struct wimax_dev * wimax_dev, enum
wimax_st new_state);
```

Arguments

`wimax_dev`

WiMAX device descriptor (properly referenced)

`new_state`

New state to switch to

Description

This implements the state changes for the wimax devices. It will

- verify that the state transition is legal (for now it'll just print a warning if not) according to the table in linux/wimax.h's documentation for 'enum wimax_st'.

- perform the actions needed for leaving the current state and whichever are needed for entering the new state.

- issue a report to user space indicating the new state (and an optional payload with information about the new state).

NOTE

wimax_dev must be locked

wimax_state_get

LINUX

Kernel Hackers ManualNovember 2026

Name

wimax_state_get — Return the current state of a WiMAX device

Synopsis

```
enum wimax_st wimax_state_get (struct wimax_dev * wimax_dev);
```

Arguments

wimax_dev

WiMAX device descriptor

Returns

Current state of the device according to its driver.

wimax_dev_init

LINUX

Kernel Hackers ManualNovember 2026

Name

`wimax_dev_init` — initialize a newly allocated instance

Synopsis

```
void wimax_dev_init (struct wimax_dev * wimax_dev);
```

Arguments

`wimax_dev`

WiMAX device descriptor to initialize.

Description

Initializes fields of a freshly allocated *`wimax_dev`* instance. This function assumes that after allocation, the memory occupied by *`wimax_dev`* was zeroed.

wimax_dev_add

LINUX

Kernel Hackers ManualNovember 2026

Name

`wimax_dev_add` — Register a new WiMAX device

Synopsis

```
int wimax_dev_add (struct wimax_dev * wimax_dev, struct
net_device * net_dev);
```

Arguments

`wimax_dev`

WiMAX device descriptor (as embedded in your *`net_dev`*'s priv data). You must have called `wimax_dev_init` on it before.

`net_dev`

net device the *`wimax_dev`* is associated with. The function expects `SET_NETDEV_DEV` and `register_netdev` were already called on it.

Description

Registers the new WiMAX device, sets up the user-kernel control interface (generic netlink) and common WiMAX infrastructure.

Note that the parts that will allow interaction with user space are setup at the very end, when the rest is in place, as once that happens, the driver might get user space control requests via netlink or from debugfs that might translate into calls into wimax_dev->op_*().

wimax_dev_rm

LINUX

Kernel Hackers ManualNovember 2026

Name

wimax_dev_rm — Unregister an existing WiMAX device

Synopsis

```
void wimax_dev_rm (struct wimax_dev * wimax_dev);
```

Arguments

wimax_dev

WiMAX device descriptor

Description

Unregisters a WiMAX device previously registered for use with wimax_add_rm.

IMPORTANT! Must call before calling unregister_netdev.

After this function returns, you will not get any more user space control requests (via netlink or debugfs) and thus to wimax_dev->ops.

Reentrancy control is ensured by setting the state to __WIMAX_ST_QUIESCING. rfkill operations coming through wimax_*rfkill*() will be stopped by the quiescing state; ops coming from the rfkill subsystem will be stopped by the support being removed by wimax_rfkill_rm.

struct wimax_dev

LINUX

Kernel Hackers ManualNovember 2026

Name

`struct wimax_dev` — Generic WiMAX device

Synopsis

```
struct wimax_dev {
  struct net_device * net_dev;
  struct list_head id_table_node;
  struct mutex mutex;
  struct mutex mutex_reset;
  enum wimax_st state;
  int (* op_msg_from_user) (struct wimax_dev *wimax_dev,const char *,const
  int (* op_rfkill_sw_toggle) (struct wimax_dev *wimax_dev,enum wimax_rf_
  int (* op_reset) (struct wimax_dev *wimax_dev);
  struct rfkill * rfkill;
  unsigned rf_hw;
  unsigned rf_sw;
  char name[32];
  struct dentry * debugfs_dentry;
};
```

Members

net_dev

[fill] Pointer to the struct net_device this WiMAX device implements.

id_table_node

[private] link to the list of wimax devices kept by id-table.c. Protected by it's own spinlock.

mutex

[private] Serializes all concurrent access and execution of operations.

mutex_reset

[private] Serializes reset operations. Needs to be a different mutex because as part of the reset operation, the driver has to call back into the stack to do things such as state change, that require wimax_dev->mutex.

state

[private] Current state of the WiMAX device.

op_msg_from_user

[fill] Driver-specific operation to handle a raw message from user space to the driver. The driver can send messages to user space using with `wimax_msg_to_user`.

op_rfkill_sw_toggle

[fill] Driver-specific operation to act on userspace (or any other agent) requesting the WiMAX device to change the RF Kill software switch (WIMAX_RF_ON or WIMAX_RF_OFF). If such hardware support is not present, it is assumed the radio cannot be switched off and it is always on (and the stack will error out when trying to switch it off). In such case, this function pointer can be left as NULL.

op_reset

[fill] Driver specific operation to reset the device. This operation should always attempt first a warm reset that does not disconnect the device from the bus and return 0. If that fails, it should resort to some sort of cold or bus reset (even if it implies a bus disconnection and device disappearance). In that case, -ENODEV should be returned to indicate the device is gone. This operation has to be synchronous, and return only when the reset is complete. In case of having had to resort to bus/cold reset implying a device disconnection, the call is allowed to return inmediately.

rfkill

[private] integration into the RF-Kill infrastructure.

rf_hw

[private] State of the hardware radio switch (OFF/ON)

rf_sw

[private] State of the software radio switch (OFF/ON)

name[32]

[fill] A way to identify this device. We need to register a name with many subsystems (rfkill, workqueue creation, etc). We can't use the network device name as that might change and in some instances we don't know it yet (until we don't call `register_netdev`). So we generate an unique one using the driver name and device bus id, place it here and use it across the board. Recommended naming: DRIVERNAME-BUSNAME:BUSID (dev->bus->name, dev->bus_id).

debugfs_dentry

[private] Used to hook up a debugfs entry. This shows up in the debugfs root as wimax\:DEVICENAME.

NOTE

wimax_dev->mutex is NOT locked when this op is being called; however, wimax_dev->mutex_reset IS locked to ensure serialization of calls to `wimax_reset`. See `wimax_reset`'s documentation.

Description

This structure defines a common interface to access all WiMAX devices from different vendors and provides a common API as well as a free-form device-specific messaging channel.

Usage

1. Embed a struct wimax_dev at *the beginning* the network device structure so that `netdev_priv` points to it.

2. `memset` it to zero

3. Initialize with `wimax_dev_init`. This will leave the WiMAX device in the `__WIMAX_ST_NULL` state.

4. Fill all the fields marked with [fill]; once called `wimax_dev_add`, those fields CANNOT be modified.

5. Call `wimax_dev_add` *after* registering the network device. This will leave the WiMAX device in the `WIMAX_ST_DOWN` state. Protect the driver's net_device->`open` against succeeding if the wimax device state is lower than `WIMAX_ST_DOWN`.

6. Select when the device is going to be turned on/initialized; for example, it could be initialized on 'ifconfig up' (when the netdev op '`open`' is called on the driver).

When the device is initialized (at 'ifconfig up' time, or right after calling `wimax_dev_add` from `_probe`, make sure the following steps are taken

a. Move the device to `WIMAX_ST_UNINITIALIZED`. This is needed so some API calls that shouldn't work until the device is ready can be blocked.

b. Initialize the device. Make sure to turn the SW radio switch off and move the device to state `WIMAX_ST_RADIO_OFF` when done. When just initialized, a device should be left in RADIO OFF state until user space devices to turn it on.

c. Query the device for the state of the hardware rfkill switch and call `wimax_rfkill_report_hw` and `wimax_rfkill_report_sw` as needed. See below.

`wimax_dev_rm` undoes before unregistering the network device. Once `wimax_dev_add` is called, the driver can get called on the wimax_dev->op_* function pointers

CONCURRENCY

The stack provides a mutex for each device that will disallow API calls happening concurrently; thus, op calls into the driver through the wimax_dev->op*() function pointers will always be serialized and *never* concurrent.

For locking, take wimax_dev->mutex is taken; (most) operations in the API have to check for `wimax_dev_is_ready` to return 0 before continuing (this is done internally).

REFERENCE COUNTING

The WiMAX device is reference counted by the associated network device. The only operation that can be used to reference the device is `wimax_dev_get_by_genl_info`, and the reference it acquires has to be released with dev_put(wimax_dev->net_dev).

RFKILL

At startup, both HW and SW radio switchess are assumed to be off.

At initialization time [after calling `wimax_dev_add`], have the driver query the device for the status of the software and hardware RF kill switches and call `wimax_report_rfkill_hw` and `wimax_rfkill_report_sw` to indicate their state. If any is missing, just call it to indicate it is ON (radio always on).

Whenever the driver detects a change in the state of the RF kill switches, it should call `wimax_report_rfkill_hw` or `wimax_report_rfkill_sw` to report it to the stack.

enum wimax_st

LINUX

Kernel Hackers ManualNovember 2026

Name

`enum wimax_st` — The different states of a WiMAX device

Synopsis

```
enum wimax_st {
  __WIMAX_ST_NULL,
  WIMAX_ST_DOWN,
  __WIMAX_ST_QUIESCING,
  WIMAX_ST_UNINITIALIZED,
  WIMAX_ST_RADIO_OFF,
  WIMAX_ST_READY,
  WIMAX_ST_SCANNING,
  WIMAX_ST_CONNECTING,
  WIMAX_ST_CONNECTED,
  __WIMAX_ST_INVALID
};
```

Constants

__WIMAX_ST_NULL

The device structure has been allocated and zeroed, but still `wimax_dev_add` hasn't been called. There is no state.

WIMAX_ST_DOWN

The device has been registered with the WiMAX and networking stacks, but it is not initialized (normally that is done with 'ifconfig DEV up' [or equivalent], which can upload firmware and enable communications with the device). In this state, the device is powered down and using as less power as possible. This state is the default after a call to `wimax_dev_add`. It is ok to have drivers move directly to `WIMAX_ST_UNINITIALIZED` or `WIMAX_ST_RADIO_OFF` in `_probe` after the call to `wimax_dev_add`. It is recommended that the driver leaves this state when calling 'ifconfig DEV up' and enters it back on 'ifconfig DEV down'.

__WIMAX_ST_QUIESCING

The device is being torn down, so no API operations are allowed to proceed except the ones needed to complete the device clean up process.

WIMAX_ST_UNINITIALIZED

[optional] Communication with the device is setup, but the device still requires some configuration before being operational. Some WiMAX API calls might work.

WIMAX_ST_RADIO_OFF

The device is fully up; radio is off (wether by hardware or software switches). It is recommended to always leave the device in this state after initialization.

WIMAX_ST_READY

The device is fully up and radio is on.

WIMAX_ST_SCANNING

[optional] The device has been instructed to scan. In this state, the device cannot be actively connected to a network.

WIMAX_ST_CONNECTING

The device is connecting to a network. This state exists because in some devices, the connect process can include a number of negotiations between user space, kernel space and the device. User space needs to know what the

device is doing. If the connect sequence in a device is atomic and fast, the device can transition directly to CONNECTED

WIMAX_ST_CONNECTED

The device is connected to a network.

__WIMAX_ST_INVALID

This is an invalid state used to mark the maximum numeric value of states.

Description

Transitions from one state to another one are atomic and can only be caused in kernel space with `wimax_state_change`. To read the state, use `wimax_state_get`.

States starting with __ are internal and shall not be used or referred to by drivers or userspace. They look ugly, but that's the point -- if any use is made non-internal to the stack, it is easier to catch on review.

All API operations [with well defined exceptions] will take the device mutex before starting and then check the state. If the state is `__WIMAX_ST_NULL`, `WIMAX_ST_DOWN`, `WIMAX_ST_UNINITIALIZED` or `__WIMAX_ST_QUIESCING`, it will drop the lock and quit with `-EINVAL`, `-ENOMEDIUM`, `-ENOTCONN` or `-ESHUTDOWN`.

The order of the definitions is important, so we can do numerical comparisons (eg: < `WIMAX_ST_RADIO_OFF` means the device is not ready to operate).

Chapter 2. Network device support

2.1. Driver Support

dev_add_pack

LINUX

Kernel Hackers ManualNovember 2026

Name

`dev_add_pack` — add packet handler

Synopsis

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

Arguments

pt

packet type declaration

Description

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

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

__dev_remove_pack

LINUX

Kernel Hackers ManualNovember 2026

Name

__dev_remove_pack — remove packet handler

Synopsis

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

Arguments

pt

packet type declaration

Description

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

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

dev_remove_pack

LINUX

Name

dev_remove_pack — remove packet handler

Synopsis

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

Arguments

pt

packet type declaration

Description

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

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

netdev_boot_setup_check

LINUX

Name

netdev_boot_setup_check — check boot time settings

Synopsis

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

Arguments

dev

the netdevice

Description

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

__dev_get_by_name

LINUX

Kernel Hackers ManualNovember 2026

Name

__dev_get_by_name — find a device by its name

Synopsis

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

Arguments

net

the applicable net namespace

name

name to find

Description

Find an interface by name. Must be called under RTNL semaphore or *dev_base_lock*. If the name is found a pointer to the device is returned. If the name is not found then NULL is returned. The reference counters are not incremented so the caller must be careful with locks.

dev_get_by_name_rcu

LINUX

Kernel Hackers ManualNovember 2026

Name

dev_get_by_name_rcu — find a device by its name

Synopsis

```
struct net_device * dev_get_by_name_rcu (struct net * net,
const char * name);
```

Arguments

net

the applicable net namespace

name

name to find

Description

Find an interface by name. If the name is found a pointer to the device is returned. If the name is not found then NULL is returned. The reference counters are not incremented so the caller must be careful with locks. The caller must hold RCU lock.

dev_get_by_name

LINUX

Kernel Hackers ManualNovember 2026

Name

dev_get_by_name — find a device by its name

Synopsis

```
struct net_device * dev_get_by_name (struct net * net, const
char * name);
```

Arguments

net

the applicable net namespace

name

name to find

Description

Find an interface by name. This can be called from any context and does its own locking. The returned handle has the usage count incremented and the caller must use `dev_put` to release it when it is no longer needed. `NULL` is returned if no matching device is found.

__dev_get_by_index

LINUX

Kernel Hackers ManualNovember 2026

Name

`__dev_get_by_index` — find a device by its ifindex

Synopsis

```
struct net_device * __dev_get_by_index (struct net * net, int
ifindex);
```

Arguments

net

the applicable net namespace

ifindex

index of device

Description

Search for an interface by index. Returns NULL if the device is not found or a pointer to the device. The device has not had its reference counter increased so the caller must be careful about locking. The caller must hold either the RTNL semaphore or *dev_base_lock*.

dev_get_by_index_rcu

LINUX

Kernel Hackers ManualNovember 2026

Name

dev_get_by_index_rcu — find a device by its ifindex

Synopsis

```
struct net_device * dev_get_by_index_rcu (struct net * net,
int ifindex);
```

Arguments

net

the applicable net namespace

ifindex

index of device

Description

Search for an interface by index. Returns NULL if the device is not found or a pointer to the device. The device has not had its reference counter increased so the caller must be careful about locking. The caller must hold RCU lock.

dev_get_by_index

LINUX

Kernel Hackers ManualNovember 2026

Name

dev_get_by_index — find a device by its ifindex

Synopsis

```
struct net_device * dev_get_by_index (struct net * net, int
ifindex);
```

Arguments

net

the applicable net namespace

ifindex

index of device

Description

Search for an interface by index. Returns NULL if the device is not found or a pointer to the device. The device returned has had a reference added and the pointer is safe until the user calls dev_put to indicate they have finished with it.

dev_getbyhwaddr_rcu

LINUX

Kernel Hackers ManualNovember 2026

Name

dev_getbyhwaddr_rcu — find a device by its hardware address

Synopsis

```
struct net_device * dev_getbyhwaddr_rcu (struct net * net,
unsigned short type, const char * ha);
```

Arguments

net

the applicable net namespace

type

media type of device

ha

hardware address

Description

Search for an interface by MAC address. Returns NULL if the device is not found or a pointer to the device. The caller must hold RCU or RTNL. The returned device has not had its ref count increased and the caller must therefore be careful about locking

dev_get_by_flags_rcu

LINUX

Kernel Hackers ManualNovember 2026

Name

`dev_get_by_flags_rcu` — find any device with given flags

Synopsis

```
struct net_device * dev_get_by_flags_rcu (struct net * net,
unsigned short if_flags, unsigned short mask);
```

Arguments

net

the applicable net namespace

if_flags

IFF_* values

mask

bitmask of bits in if_flags to check

Description

Search for any interface with the given flags. Returns NULL if a device is not found or a pointer to the device. Must be called inside `rcu_read_lock`, and result refcount is unchanged.

dev_valid_name

LINUX

Kernel Hackers ManualNovember 2026

Name

`dev_valid_name` — check if name is okay for network device

Synopsis

```
int dev_valid_name (const char * name);
```

Arguments

name

name string

Description

Network device names need to be valid file names to to allow sysfs to work. We also disallow any kind of whitespace.

dev_alloc_name

LINUX

Kernel Hackers ManualNovember 2026

Name

`dev_alloc_name` — allocate a name for a device

Synopsis

```
int dev_alloc_name (struct net_device * dev, const char *
name);
```

Arguments

dev

device

name

name format string

Description

Passed a format string - eg “ltd” it will try and find a suitable id. It scans list of devices to build up a free map, then chooses the first empty slot. The caller must hold the dev_base or rtnl lock while allocating the name and adding the device in order to avoid duplicates. Limited to bits_per_byte * page size devices (ie 32K on most platforms). Returns the number of the unit assigned or a negative errno code.

netdev_features_change

LINUX

Kernel Hackers ManualNovember 2026

Name

`netdev_features_change` — device changes features

Synopsis

```
void netdev_features_change (struct net_device * dev);
```

Arguments

dev

device to cause notification

Description

Called to indicate a device has changed features.

netdev_state_change

LINUX

Kernel Hackers ManualNovember 2026

Name

`netdev_state_change` — device changes state

Synopsis

```
void netdev_state_change (struct net_device * dev);
```

Arguments

dev

device to cause notification

Description

Called to indicate a device has changed state. This function calls the notifier chains for netdev_chain and sends a NEWLINK message to the routing socket.

dev_load

LINUX

Name

`dev_load` — load a network module

Synopsis

```
void dev_load (struct net * net, const char * name);
```

Arguments

net

the applicable net namespace

name

name of interface

Description

If a network interface is not present and the process has suitable privileges this function loads the module. If module loading is not available in this kernel then it becomes a nop.

dev_open

LINUX

Name

dev_open — prepare an interface for use.

Synopsis

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

Arguments

dev

device to open

Description

Takes a device from down to up state. The device's private open function is invoked and then the multicast lists are loaded. Finally the device is moved into the up state and a NETDEV_UP message is sent to the netdev notifier chain.

Calling this function on an active interface is a nop. On a failure a negative errno code is returned.

dev_close

LINUX

Name

dev_close — shutdown an interface.

Synopsis

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

Arguments

dev

device to shutdown

Description

This function moves an active device into down state. A NETDEV_GOING_DOWN is sent to the netdev notifier chain. The device is then deactivated and finally a NETDEV_DOWN is sent to the notifier chain.

dev_disable_lro

LINUX

Kernel Hackers ManualNovember 2026

Name

dev_disable_lro — disable Large Receive Offload on a device

Synopsis

```
void dev_disable_lro (struct net_device * dev);
```

Arguments

dev

device

Description

Disable Large Receive Offload (LRO) on a net device. Must be called under RTNL. This is needed if received packets may be forwarded to another interface.

register_netdevice_notifier

LINUX

Kernel Hackers ManualNovember 2026

Name

`register_netdevice_notifier` — register a network notifier block

Synopsis

```
int register_netdevice_notifier (struct notifier_block * nb);
```

Arguments

nb

notifier

Description

Register a notifier to be called when network device events occur. The notifier passed is linked into the kernel structures and must not be reused until it has been unregistered. A negative errno code is returned on a failure.

When registered all registration and up events are replayed to the new notifier to allow device to have a race free view of the network device list.

unregister_netdevice_notifier

LINUX

Kernel Hackers ManualNovember 2026

Name

`unregister_netdevice_notifier` — unregister a network notifier block

Synopsis

```
int unregister_netdevice_notifier (struct notifier_block *
nb);
```

Arguments

nb

notifier

Description

Unregister a notifier previously registered by `register_netdevice_notifier`. The notifier is unlinked into the kernel structures and may then be reused. A negative errno code is returned on a failure.

After unregistering unregister and down device events are synthesized for all devices on the device list to the removed notifier to remove the need for special case cleanup code.

call_netdevice_notifiers

LINUX

Kernel Hackers ManualNovember 2026

Name

`call_netdevice_notifiers` — call all network notifier blocks

Synopsis

```
int call_netdevice_notifiers (unsigned long val, struct
net_device * dev);
```

Arguments

val

value passed unmodified to notifier function

dev

net_device pointer passed unmodified to notifier function

Description

Call all network notifier blocks. Parameters and return value are as for `raw_notifier_call_chain`.

dev_forward_skb

LINUX

Kernel Hackers ManualNovember 2026

Name

`dev_forward_skb` — loopback an skb to another netif

Synopsis

```
int dev_forward_skb (struct net_device * dev, struct sk_buff *
skb);
```

Arguments

dev

destination network device

skb

buffer to forward

return values

NET_RX_SUCCESS (no congestion) NET_RX_DROP (packet was dropped, but freed)

dev_forward_skb can be used for injecting an skb from the start_xmit function of one device into the receive queue of another device.

The receiving device may be in another namespace, so we have to clear all information in the skb that could impact namespace isolation.

netif_set_real_num_rx_queues

LINUX

Kernel Hackers ManualNovember 2026

Name

`netif_set_real_num_rx_queues` — set actual number of RX queues used

Synopsis

```
int netif_set_real_num_rx_queues (struct net_device * dev,
unsigned int rxq);
```

Arguments

dev

Network device

rxq

Actual number of RX queues

Description

This must be called either with the rtnl_lock held or before registration of the net device. Returns 0 on success, or a negative error code. If called before registration, it always succeeds.

netif_device_detach

LINUX

Kernel Hackers ManualNovember 2026

Name

`netif_device_detach` — mark device as removed

Synopsis

```
void netif_device_detach (struct net_device * dev);
```

Arguments

dev

network device

Description

Mark device as removed from system and therefore no longer available.

netif_device_attach

LINUX

Kernel Hackers ManualNovember 2026

Name

`netif_device_attach` — mark device as attached

Synopsis

```
void netif_device_attach (struct net_device * dev);
```

Arguments

dev

network device

Description

Mark device as attached from system and restart if needed.

skb_gso_segment

LINUX

Kernel Hackers ManualNovember 2026

Name

skb_gso_segment — Perform segmentation on skb.

Synopsis

```
struct sk_buff * skb_gso_segment (struct sk_buff * skb, u32
features);
```

Arguments

skb

buffer to segment

features

features for the output path (see dev->features)

Description

This function segments the given skb and returns a list of segments.

It may return NULL if the skb requires no segmentation. This is only possible when GSO is used for verifying header integrity.

dev_queue_xmit

LINUX

Kernel Hackers ManualNovember 2026

Name

dev_queue_xmit — transmit a buffer

Synopsis

```
int dev_queue_xmit (struct sk_buff * skb);
```

Arguments

skb

buffer to transmit

Description

Queue a buffer for transmission to a network device. The caller must have set the device and priority and built the buffer before calling this function. The function can be called from an interrupt.

A negative errno code is returned on a failure. A success does not guarantee the frame will be transmitted as it may be dropped due to congestion or traffic shaping.

---------- I notice this method can also return errors from the queue disciplines, including NET_XMIT_DROP, which is a positive value. So, errors can also be positive.

Regardless of the return value, the skb is consumed, so it is currently difficult to retry a send to this method. (You can bump the ref count before sending to hold a reference for retry if you are careful.)

When calling this method, interrupts MUST be enabled. This is because the BH enable code must have IRQs enabled so that it will not deadlock. --BLG

rps_may_expire_flow

LINUX

Kernel Hackers ManualNovember 2026

Name

`rps_may_expire_flow` — check whether an RFS hardware filter may be removed

Synopsis

```
bool rps_may_expire_flow (struct net_device * dev, u16
rxq_index, u32 flow_id, u16 filter_id);
```

Arguments

dev

Device on which the filter was set

rxq_index

RX queue index

flow_id

Flow ID passed to `ndo_rx_flow_steer`

filter_id

Filter ID returned by `ndo_rx_flow_steer`

Description

Drivers that implement `ndo_rx_flow_steer` should periodically call this function for each installed filter and remove the filters for which it returns `true`.

netif_rx

LINUX

Kernel Hackers ManualNovember 2026

Name

`netif_rx` — post buffer to the network code

Synopsis

```
int netif_rx (struct sk_buff * skb);
```

Arguments

skb

buffer to post

Description

This function receives a packet from a device driver and queues it for the upper (protocol) levels to process. It always succeeds. The buffer may be dropped during processing for congestion control or by the protocol layers.

return values

NET_RX_SUCCESS (no congestion) NET_RX_DROP (packet was dropped)

netdev_rx_handler_register

LINUX

Kernel Hackers ManualNovember 2026

Name

`netdev_rx_handler_register` — register receive handler

Synopsis

```
int netdev_rx_handler_register (struct net_device * dev,
rx_handler_func_t * rx_handler, void * rx_handler_data);
```

Arguments

dev

device to register a handler for

rx_handler

receive handler to register

rx_handler_data

data pointer that is used by rx handler

Description

Register a receive hander for a device. This handler will then be called from __netif_receive_skb. A negative errno code is returned on a failure.

The caller must hold the rtnl_mutex.

For a general description of rx_handler, see enum rx_handler_result.

netdev_rx_handler_unregister

LINUX

Kernel Hackers ManualNovember 2026

Name

netdev_rx_handler_unregister — unregister receive handler

Synopsis

```
void netdev_rx_handler_unregister (struct net_device * dev);
```

Arguments

dev

device to unregister a handler from

Description

Unregister a receive hander from a device.

The caller must hold the rtnl_mutex.

netif_receive_skb

LINUX

Kernel Hackers ManualNovember 2026

Name

`netif_receive_skb` — process receive buffer from network

Synopsis

```
int netif_receive_skb (struct sk_buff * skb);
```

Arguments

skb

buffer to process

Description

`netif_receive_skb` is the main receive data processing function. It always succeeds. The buffer may be dropped during processing for congestion control or by the protocol layers.

This function may only be called from softirq context and interrupts should be enabled.

Return values (usually ignored):

NET_RX_SUCCESS

no congestion

NET_RX_DROP

packet was dropped

__napi_schedule

LINUX

Kernel Hackers ManualNovember 2026

Name

`__napi_schedule` — schedule for receive

Synopsis

```
void __napi_schedule (struct napi_struct * n);
```

Arguments

n

entry to schedule

Description

The entry's receive function will be scheduled to run. Consider using `__napi_schedule_irqoff` if hard irqs are masked.

__napi_schedule_irqoff

LINUX

Kernel Hackers ManualNovember 2026

Name

`__napi_schedule_irqoff` — schedule for receive

Synopsis

```
void __napi_schedule_irqoff (struct napi_struct * n);
```

Arguments

n

entry to schedule

Description

Variant of `__napi_schedule` assuming hard irqs are masked

register_gifconf

LINUX

Kernel Hackers ManualNovember 2026

Name

`register_gifconf` — register a SIOCGIF handler

Synopsis

```
int register_gifconf (unsigned int family, gifconf_func_t *
gifconf);
```

Arguments

family

Address family

gifconf

Function handler

Description

Register protocol dependent address dumping routines. The handler that is passed must not be freed or reused until it has been replaced by another handler.

netdev_set_master

LINUX

Kernel Hackers ManualNovember 2026

Name

`netdev_set_master` — set up master pointer

Synopsis

```
int netdev_set_master (struct net_device * slave, struct
net_device * master);
```

Arguments

`slave`

slave device

`master`

new master device

Description

Changes the master device of the slave. Pass `NULL` to break the bonding. The caller must hold the RTNL semaphore. On a failure a negative errno code is returned. On success the reference counts are adjusted and the function returns zero.

netdev_set_bond_master

LINUX

Kernel Hackers ManualNovember 2026

Name

`netdev_set_bond_master` — set up bonding master/slave pair

Synopsis

```
int netdev_set_bond_master (struct net_device * slave, struct
net_device * master);
```

Arguments

slave

slave device

master

new master device

Description

Changes the master device of the slave. Pass `NULL` to break the bonding. The caller must hold the RTNL semaphore. On a failure a negative errno code is returned. On success `RTM_NEWLINK` is sent to the routing socket and the function returns zero.

dev_set_promiscuity

LINUX

Kernel Hackers ManualNovember 2026

Name

dev_set_promiscuity — update promiscuity count on a device

Synopsis

```
int dev_set_promiscuity (struct net_device * dev, int inc);
```

Arguments

dev

device

inc

modifier

Description

Add or remove promiscuity from a device. While the count in the device remains above zero the interface remains promiscuous. Once it hits zero the device reverts back to normal filtering operation. A negative inc value is used to drop promiscuity on the device. Return 0 if successful or a negative errno code on error.

dev_set_allmulti

LINUX

Name

`dev_set_allmulti` — update allmulti count on a device

Synopsis

```
int dev_set_allmulti (struct net_device * dev, int inc);
```

Arguments

dev

device

inc

modifier

Description

Add or remove reception of all multicast frames to a device. While the count in the device remains above zero the interface remains listening to all interfaces. Once it hits zero the device reverts back to normal filtering operation. A negative *inc* value is used to drop the counter when releasing a resource needing all multicasts. Return 0 if successful or a negative errno code on error.

dev_ethtool_get_settings

LINUX

Name

`dev_ethtool_get_settings` — call device's ethtool_ops::`get_settings`

Synopsis

```
int dev_ethtool_get_settings (struct net_device * dev, struct
ethtool_cmd * cmd);
```

Arguments

dev

device

cmd

memory area for ethtool_ops::`get_settings` result

Description

The cmd arg is initialized properly (cleared and

ethtool_cmd

:cmd field set to ETHTOOL_GSET).

Return device's ethtool_ops::`get_settings` result value or -EOPNOTSUPP when device doesn't expose

ethtool_ops

:`get_settings` operation.

dev_get_flags

LINUX

Kernel Hackers ManualNovember 2026

Name

dev_get_flags — get flags reported to userspace

Synopsis

```
unsigned dev_get_flags (const struct net_device * dev);
```

Arguments

dev

device

Description

Get the combination of flag bits exported through APIs to userspace.

dev_change_flags

LINUX

Kernel Hackers ManualNovember 2026

Name

dev_change_flags — change device settings

Synopsis

```
int dev_change_flags (struct net_device * dev, unsigned
flags);
```

Arguments

dev

device

flags

device state flags

Description

Change settings on device based state flags. The flags are in the userspace exported format.

dev_set_mtu

LINUX

Kernel Hackers ManualNovember 2026

Name

`dev_set_mtu` — Change maximum transfer unit

Synopsis

```
int dev_set_mtu (struct net_device * dev, int new_mtu);
```

Arguments

dev

device

new_mtu

new transfer unit

Description

Change the maximum transfer size of the network device.

dev_set_group

LINUX

Kernel Hackers ManualNovember 2026

Name

dev_set_group — Change group this device belongs to

Synopsis

```
void dev_set_group (struct net_device * dev, int new_group);
```

Arguments

dev

device

new_group

group this device should belong to

dev_set_mac_address

LINUX

Kernel Hackers ManualNovember 2026

Name

`dev_set_mac_address` — Change Media Access Control Address

Synopsis

```
int dev_set_mac_address (struct net_device * dev, struct
sockaddr * sa);
```

Arguments

dev

device

sa

new address

Description

Change the hardware (MAC) address of the device

dev_get_phys_port_id

LINUX

Kernel Hackers ManualNovember 2026

Name

`dev_get_phys_port_id` — Get device physical port ID

Synopsis

```
int dev_get_phys_port_id (struct net_device * dev, struct
netdev_phys_port_id * ppid);
```

Arguments

dev

device

ppid

port ID

Description

Get device physical port ID

netdev_update_features

LINUX

Kernel Hackers ManualNovember 2026

Name

`netdev_update_features` — recalculate device features

Synopsis

```
void netdev_update_features (struct net_device * dev);
```

Arguments

dev

the device to check

Description

Recalculate dev->features set and send notifications if it has changed. Should be called after driver or hardware dependent conditions might have changed that influence the features.

netdev_change_features

LINUX

Name

`netdev_change_features` — recalculate device features

Synopsis

```
void netdev_change_features (struct net_device * dev);
```

Arguments

`dev`

the device to check

Description

Recalculate dev->features set and send notifications even if they have not changed. Should be called instead of `netdev_update_features` if also dev->vlan_features might have changed to allow the changes to be propagated to stacked VLAN devices.

netif_stacked_transfer_operstate

LINUX

Kernel Hackers ManualNovember 2026

Name

`netif_stacked_transfer_operstate` — transfer operstate

Synopsis

```
void netif_stacked_transfer_operstate (const struct net_device
* rootdev, struct net_device * dev);
```

Arguments

rootdev

the root or lower level device to transfer state from

dev

the device to transfer operstate to

Description

Transfer operational state from root to device. This is normally called when a stacking relationship exists between the root device and the device(a leaf device).

register_netdevice

LINUX

Kernel Hackers ManualNovember 2026

Name

`register_netdevice` — register a network device

Synopsis

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

Arguments

dev

device to register

Description

Take a completed network device structure and add it to the kernel interfaces. A NETDEV_REGISTER message is sent to the netdev notifier chain. 0 is returned on success. A negative errno code is returned on a failure to set up the device, or if the name is a duplicate.

Callers must hold the rtnl semaphore. You may want register_netdev instead of this.

BUGS

The locking appears insufficient to guarantee two parallel registers will not get the same name.

init_dummy_netdev

LINUX

Kernel Hackers ManualNovember 2026

Name

init_dummy_netdev — init a dummy network device for NAPI

Synopsis

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

Arguments

dev

device to init

Description

This takes a network device structure and initialize the minimum amount of fields so it can be used to schedule NAPI polls without registering a full blown interface. This is to be used by drivers that need to tie several hardware interfaces to a single NAPI poll scheduler due to HW limitations.

register_netdev

LINUX

Kernel Hackers ManualNovember 2026

Name

`register_netdev` — register a network device

Synopsis

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

Arguments

dev

device to register

Description

Take a completed network device structure and add it to the kernel interfaces. A NETDEV_REGISTER message is sent to the netdev notifier chain. 0 is returned on success. A negative errno code is returned on a failure to set up the device, or if the name is a duplicate.

This is a wrapper around register_netdevice that takes the rtnl semaphore and expands the device name if you passed a format string to alloc_netdev.

dev_get_stats

LINUX

Kernel Hackers ManualNovember 2026

Name

dev_get_stats — get network device statistics

Synopsis

```
struct rtnl_link_stats64 * dev_get_stats (struct net_device *
dev, struct rtnl_link_stats64 * storage);
```

Arguments

dev

device to get statistics from

storage

place to store stats

Description

Get network statistics from device. Return *storage*. The device driver may provide its own method by setting dev->netdev_ops->get_stats64 or dev->netdev_ops->get_stats; otherwise the internal statistics structure is used.

alloc_netdev_mqs

LINUX

Kernel Hackers ManualNovember 2026

Name

alloc_netdev_mqs — allocate network device

Synopsis

```
struct net_device * alloc_netdev_mqs (int sizeof_priv, const
char * name, void (*setup) (struct net_device *), unsigned int
txqs, unsigned int rxqs);
```

Arguments

sizeof_priv

size of private data to allocate space for

name

device name format string

setup

callback to initialize device

txqs

the number of TX subqueues to allocate

rxqs

the number of RX subqueues to allocate

Description

Allocates a struct net_device with private data area for driver use and performs basic initialization. Also allocates subquue structs for each queue on the device.

free_netdev

LINUX

Kernel Hackers ManualNovember 2026

Name

`free_netdev` — free network device

Synopsis

```
void free_netdev (struct net_device * dev);
```

Arguments

dev

device

Description

This function does the last stage of destroying an allocated device interface. The reference to the device object is released. If this is the last reference then it will be freed.

synchronize_net

LINUX

Kernel Hackers ManualNovember 2026

Name

`synchronize_net` — Synchronize with packet receive processing

Synopsis

```
void synchronize_net ( void);
```

Arguments

void

no arguments

Description

Wait for packets currently being received to be done. Does not block later packets from starting.

unregister_netdevice_queue

LINUX

Kernel Hackers ManualNovember 2026

Name

`unregister_netdevice_queue` — remove device from the kernel

Synopsis

```
void unregister_netdevice_queue (struct net_device * dev,
struct list_head * head);
```

Arguments

dev

device

head

list

Description

This function shuts down a device interface and removes it from the kernel tables. If head not NULL, device is queued to be unregistered later.

Callers must hold the rtnl semaphore. You may want `unregister_netdev` instead of this.

unregister_netdevice_many

LINUX

Kernel Hackers ManualNovember 2026

Name

`unregister_netdevice_many` — unregister many devices

Synopsis

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

Arguments

`head`

list of devices

unregister_netdev

LINUX

Name

`unregister_netdev` — remove device from the kernel

Synopsis

```
void unregister_netdev (struct net_device * dev);
```

Arguments

dev

device

Description

This function shuts down a device interface and removes it from the kernel tables.

This is just a wrapper for unregister_netdevice that takes the rtnl semaphore. In general you want to use this and not unregister_netdevice.

dev_change_net_namespace

LINUX

Name

`dev_change_net_namespace` — move device to different nethost namespace

Synopsis

```
int dev_change_net_namespace (struct net_device * dev, struct
net * net, const char * pat);
```

Arguments

dev

device

net

network namespace

pat

If not NULL name pattern to try if the current device name is already taken in the destination network namespace.

Description

This function shuts down a device interface and moves it to a new network namespace. On success 0 is returned, on a failure a netagive errno code is returned.

Callers must hold the rtnl semaphore.

netdev_increment_features

LINUX

Kernel Hackers ManualNovember 2026

Name

`netdev_increment_features` — increment feature set by one

Synopsis

```
u32 netdev_increment_features (u32 all, u32 one, u32 mask);
```

Arguments

all

current feature set

one

new feature set

mask

mask feature set

Description

Computes a new feature set after adding a device with feature set *one* to the master device with current feature set *all*. Will not enable anything that is off in *mask*. Returns the new feature set.

eth_header

LINUX

Kernel Hackers ManualNovember 2026

Name

eth_header — create the Ethernet header

Synopsis

```
int eth_header (struct sk_buff * skb, struct net_device * dev,
unsigned short type, const void * daddr, const void * saddr,
unsigned len);
```

Arguments

skb

buffer to alter

dev

source device

type

Ethernet type field

daddr

destination address (NULL leave destination address)

saddr

source address (NULL use device source address)

len

packet length (<= skb->len)

Description

Set the protocol type. For a packet of type ETH_P_802_3/2 we put the length in here instead.

eth_rebuild_header

LINUX

Kernel Hackers ManualNovember 2026

Name

`eth_rebuild_header` — rebuild the Ethernet MAC header.

Synopsis

```
int eth_rebuild_header (struct sk_buff * skb);
```

Arguments

`skb`

socket buffer to update

Description

This is called after an ARP or IPV6 ndisc it's resolution on this sk_buff. We now let protocol (ARP) fill in the other fields.

This routine CANNOT use cached dst->neigh! Really, it is used only when dst->neigh is wrong.

eth_type_trans

LINUX

Name

`eth_type_trans` — determine the packet's protocol ID.

Synopsis

```
__be16 eth_type_trans (struct sk_buff * skb, struct net_device
* dev);
```

Arguments

skb

received socket data

dev

receiving network device

Description

The rule here is that we assume 802.3 if the type field is short enough to be a length. This is normal practice and works for any 'now in use' protocol.

eth_header_parse

LINUX

Name

`eth_header_parse` — extract hardware address from packet

Synopsis

```
int eth_header_parse (const struct sk_buff * skb, unsigned
char * haddr);
```

Arguments

`skb`

packet to extract header from

`haddr`

destination buffer

eth_header_cache

LINUX

Kernel Hackers ManualNovember 2026

Name

`eth_header_cache` — fill cache entry from neighbour

Synopsis

```
int eth_header_cache (const struct neighbour * neigh, struct
hh_cache * hh);
```

Arguments

neigh

source neighbour

hh

destination cache entry Create an Ethernet header template from the neighbour.

eth_header_cache_update

LINUX

Kernel Hackers ManualNovember 2026

Name

eth_header_cache_update — update cache entry

Synopsis

```
void eth_header_cache_update (struct hh_cache * hh, const
struct net_device * dev, const unsigned char * haddr);
```

Arguments

hh

destination cache entry

dev

network device

haddr

new hardware address

Description

Called by Address Resolution module to notify changes in address.

eth_mac_addr

LINUX

Kernel Hackers ManualNovember 2026

Name

`eth_mac_addr` — set new Ethernet hardware address

Synopsis

```
int eth_mac_addr (struct net_device * dev, void * p);
```

Arguments

dev

network device

p

socket address Change hardware address of device.

Description

This doesn't change hardware matching, so needs to be overridden for most real devices.

eth_change_mtu

LINUX

Kernel Hackers ManualNovember 2026

Name

`eth_change_mtu` — set new MTU size

Synopsis

```
int eth_change_mtu (struct net_device * dev, int new_mtu);
```

Arguments

dev

network device

new_mtu

new Maximum Transfer Unit

Description

Allow changing MTU size. Needs to be overridden for devices supporting jumbo frames.

ether_setup

LINUX

Kernel Hackers ManualNovember 2026

Name

`ether_setup` — setup Ethernet network device

Synopsis

```
void ether_setup (struct net_device * dev);
```

Arguments

dev

network device Fill in the fields of the device structure with Ethernet-generic values.

alloc_etherdev_mqs

LINUX

Kernel Hackers ManualNovember 2026

Name

`alloc_etherdev_mqs` — Allocates and sets up an Ethernet device

Synopsis

```
struct net_device * alloc_etherdev_mqs (int sizeof_priv,
unsigned int txqs, unsigned int rxqs);
```

Arguments

sizeof_priv

Size of additional driver-private structure to be allocated for this Ethernet device

txqs

The number of TX queues this device has.

rxqs

The number of RX queues this device has.

Description

Fill in the fields of the device structure with Ethernet-generic values. Basically does everything except registering the device.

Constructs a new net device, complete with a private data area of size (sizeof_priv). A 32-byte (not bit) alignment is enforced for this private data area.

netif_carrier_on

LINUX

Kernel Hackers ManualNovember 2026

Name

`netif_carrier_on` — set carrier

Synopsis

```
void netif_carrier_on (struct net_device * dev);
```

Arguments

dev

network device

Description

Device has detected that carrier.

netif_carrier_off

LINUX

Kernel Hackers ManualNovember 2026

Name

`netif_carrier_off` — clear carrier

Synopsis

```
void netif_carrier_off (struct net_device * dev);
```

Arguments

dev

network device

Description

Device has detected loss of carrier.

netif_notify_peers

LINUX

Kernel Hackers ManualNovember 2026

Name

`netif_notify_peers` — notify network peers about existence of *dev*

Synopsis

```
void netif_notify_peers (struct net_device * dev);
```

Arguments

dev

network device

Description

Generate traffic such that interested network peers are aware of *dev*, such as by generating a gratuitous ARP. This may be used when a device wants to inform the rest of the network about some sort of reconfiguration such as a failover event or virtual machine migration.

is_link_local_ether_addr

LINUX

Kernel Hackers ManualNovember 2026

Name

is_link_local_ether_addr — Determine if given Ethernet address is link-local

Synopsis

```
bool is_link_local_ether_addr (const u8 * addr);
```

Arguments

addr

Pointer to a six-byte array containing the Ethernet address

Description

Return true if address is link local reserved addr (01:80:c2:00:00:0X) per IEEE 802.1Q 8.6.3 Frame filtering.

is_zero_ether_addr

LINUX

Kernel Hackers ManualNovember 2026

Name

`is_zero_ether_addr` — Determine if give Ethernet address is all zeros.

Synopsis

```
int is_zero_ether_addr (const u8 * addr);
```

Arguments

addr

Pointer to a six-byte array containing the Ethernet address

Description

Return true if the address is all zeroes.

is_multicast_ether_addr

LINUX

Kernel Hackers ManualNovember 2026

Name

`is_multicast_ether_addr` — Determine if the Ethernet address is a multicast.

Synopsis

```
int is_multicast_ether_addr (const u8 * addr);
```

Arguments

addr

Pointer to a six-byte array containing the Ethernet address

Description

Return true if the address is a multicast address. By definition the broadcast address is also a multicast address.

is_local_ether_addr

LINUX

Name

`is_local_ether_addr` — Determine if the Ethernet address is locally-assigned one (IEEE 802).

Synopsis

```
int is_local_ether_addr (const u8 * addr);
```

Arguments

addr

Pointer to a six-byte array containing the Ethernet address

Description

Return true if the address is a local address.

is_broadcast_ether_addr

LINUX

Name

`is_broadcast_ether_addr` — Determine if the Ethernet address is broadcast

Synopsis

```
int is_broadcast_ether_addr (const u8 * addr);
```

Arguments

addr

Pointer to a six-byte array containing the Ethernet address

Description

Return true if the address is the broadcast address.

is_unicast_ether_addr

LINUX

Kernel Hackers ManualNovember 2026

Name

`is_unicast_ether_addr` — Determine if the Ethernet address is unicast

Synopsis

```
int is_unicast_ether_addr (const u8 * addr);
```

Arguments

addr

Pointer to a six-byte array containing the Ethernet address

Description

Return true if the address is a unicast address.

is_valid_ether_addr

LINUX

Kernel Hackers ManualNovember 2026

Name

`is_valid_ether_addr` — Determine if the given Ethernet address is valid

Synopsis

```
int is_valid_ether_addr (const u8 * addr);
```

Arguments

addr

Pointer to a six-byte array containing the Ethernet address

Description

Check that the Ethernet address (MAC) is not 00:00:00:00:00:00, is not a multicast address, and is not FF:FF:FF:FF:FF:FF.

Return true if the address is valid.

eth_random_addr

LINUX

Kernel Hackers ManualNovember 2026

Name

`eth_random_addr` — Generate software assigned random Ethernet address

Synopsis

```
void eth_random_addr (u8 * addr);
```

Arguments

addr

Pointer to a six-byte array containing the Ethernet address

Description

Generate a random Ethernet address (MAC) that is not multicast and has the local assigned bit set.

eth_broadcast_addr

LINUX

Kernel Hackers ManualNovember 2026

Name

`eth_broadcast_addr` — Assign broadcast address

Synopsis

```
void eth_broadcast_addr (u8 * addr);
```

Arguments

addr

Pointer to a six-byte array containing the Ethernet address

Description

Assign the broadcast address to the given address array.

eth_zero_addr

LINUX

Kernel Hackers ManualNovember 2026

Name

`eth_zero_addr` — Assign zero address

Synopsis

```
void eth_zero_addr (u8 * addr);
```

Arguments

addr

Pointer to a six-byte array containing the Ethernet address

Description

Assign the zero address to the given address array.

eth_hw_addr_random

LINUX

Kernel Hackers ManualNovember 2026

Name

eth_hw_addr_random — Generate software assigned random Ethernet and set device flag

Synopsis

```
void eth_hw_addr_random (struct net_device * dev);
```

Arguments

dev

pointer to net_device structure

Description

Generate a random Ethernet address (MAC) to be used by a net device and set addr_assign_type so the state can be read by sysfs and be used by userspace.

ether_addr_copy

LINUX

Kernel Hackers ManualNovember 2026

Name

`ether_addr_copy` — Copy an Ethernet address

Synopsis

```
void ether_addr_copy (u8 * dst, const u8 * src);
```

Arguments

dst

Pointer to a six-byte array Ethernet address destination

src

Pointer to a six-byte array Ethernet address source

Please note

dst & src must both be aligned to u16.

compare_ether_addr

LINUX

Kernel Hackers ManualNovember 2026

Name

`compare_ether_addr` — Compare two Ethernet addresses

Synopsis

```
unsigned compare_ether_addr (const u8 * addr1, const u8 *
addr2);
```

Arguments

addr1

Pointer to a six-byte array containing the Ethernet address

addr2

Pointer other six-byte array containing the Ethernet address

Description

Compare two ethernet addresses, returns 0 if equal

ether_addr_equal

LINUX

Kernel Hackers ManualNovember 2026

Name

`ether_addr_equal` — Compare two Ethernet addresses

Synopsis

```
bool ether_addr_equal (const u8 * addr1, const u8 * addr2);
```

Arguments

addr1

Pointer to a six-byte array containing the Ethernet address

addr2

Pointer other six-byte array containing the Ethernet address

Description

Compare two ethernet addresses, returns true if equal

compare_ether_addr_64bits

LINUX

Name

`compare_ether_addr_64bits` — Compare two Ethernet addresses

Synopsis

```
unsigned compare_ether_addr_64bits (const u8 addr1[6+2], const
u8 addr2[6+2]);
```

Arguments

`addr1[6+2]`

Pointer to an array of 8 bytes

`addr2[6+2]`

Pointer to an other array of 8 bytes

Description

Compare two ethernet addresses, returns 0 if equal. Same result than “memcmp(addr1, addr2, ETH_ALEN)” but without conditional branches, and possibly long word memory accesses on CPU allowing cheap unaligned memory reads. arrays = { byte1, byte2, byte3, byte4, byte6, byte7, pad1, pad2}

Please note that alignment of addr1 & addr2 is only guaranted to be 16 bits.

ether_addr_equal_unaligned

LINUX

Name

`ether_addr_equal_unaligned` — Compare two not u16 aligned Ethernet addresses

Synopsis

```
bool ether_addr_equal_unaligned (const u8 * addr1, const u8 * addr2);
```

Arguments

addr1

Pointer to a six-byte array containing the Ethernet address

addr2

Pointer other six-byte array containing the Ethernet address

Description

Compare two Ethernet addresses, returns true if equal

Please note

Use only when any Ethernet address may not be u16 aligned.

is_etherdev_addr

LINUX

Name

`is_etherdev_addr` — Tell if given Ethernet address belongs to the device.

Synopsis

```
bool is_etherdev_addr (const struct net_device * dev, const u8
addr[6 + 2]);
```

Arguments

dev

Pointer to a device structure

addr[6 + 2]

Pointer to a six-byte array containing the Ethernet address

Description

Compare passed address with all addresses of the device. Return true if the address if one of the device addresses.

Note that this function calls `compare_ether_addr_64bits` so take care of the right padding.

compare_ether_header

LINUX

Name

`compare_ether_header` — Compare two Ethernet headers

Synopsis

```
unsigned long compare_ether_header (const void * a, const void
* b);
```

Arguments

a

Pointer to Ethernet header

b

Pointer to Ethernet header

Description

Compare two ethernet headers, returns 0 if equal. This assumes that the network header (i.e., IP header) is 4-byte aligned OR the platform can handle unaligned access. This is the case for all packets coming into netif_receive_skb or similar entry points.

eth_skb_pad

LINUX

Name

`eth_skb_pad` — Pad buffer to mininum number of octets for Ethernet frame

Synopsis

```
int eth_skb_pad (struct sk_buff * skb);
```

Arguments

skb

Buffer to pad

Description

An Ethernet frame should have a minimum size of 60 bytes. This function takes short frames and pads them with zeros up to the 60 byte limit.

napi_schedule_prep

LINUX

Name

`napi_schedule_prep` — check if napi can be scheduled

Synopsis

```
int napi_schedule_prep (struct napi_struct * n);
```

Arguments

n

napi context

Description

Test if NAPI routine is already running, and if not mark it as running. This is used as a condition variable insure only one NAPI poll instance runs. We also make sure there is no pending NAPI disable.

napi_schedule

LINUX

Kernel Hackers ManualNovember 2026

Name

`napi_schedule` — schedule NAPI poll

Synopsis

```
void napi_schedule (struct napi_struct * n);
```

Arguments

n

napi context

Description

Schedule NAPI poll routine to be called if it is not already running.

napi_schedule_irqoff

LINUX

Kernel Hackers ManualNovember 2026

Name

`napi_schedule_irqoff` — schedule NAPI poll

Synopsis

```
void napi_schedule_irqoff (struct napi_struct * n);
```

Arguments

n

napi context

Description

Variant of `napi_schedule`, assuming hard irqs are masked.

napi_disable

LINUX

Kernel Hackers ManualNovember 2026

Name

`napi_disable` — prevent NAPI from scheduling

Synopsis

```
void napi_disable (struct napi_struct * n);
```

Arguments

n

napi context

Description

Stop NAPI from being scheduled on this context. Waits till any outstanding processing completes.

napi_enable

LINUX

Name

`napi_enable` — enable NAPI scheduling

Synopsis

```
void napi_enable (struct napi_struct * n);
```

Arguments

n

napi context

Description

Resume NAPI from being scheduled on this context. Must be paired with napi_disable.

napi_synchronize

LINUX

Name

`napi_synchronize` — wait until NAPI is not running

Synopsis

```
void napi_synchronize (const struct napi_struct * n);
```

Arguments

n

napi context

Description

Wait until NAPI is done being scheduled on this context. Waits till any outstanding processing completes but does not disable future activations.

netdev_priv

LINUX

Kernel Hackers ManualNovember 2026

Name

`netdev_priv` — access network device private data

Synopsis

```
void * netdev_priv (const struct net_device * dev);
```

Arguments

dev

network device

Description

Get network device private data

netif_start_queue

LINUX

Kernel Hackers ManualNovember 2026

Name

`netif_start_queue` — allow transmit

Synopsis

```
void netif_start_queue (struct net_device * dev);
```

Arguments

dev

network device

Description

Allow upper layers to call the device hard_start_xmit routine.

netif_wake_queue

LINUX

Kernel Hackers ManualNovember 2026

Name

`netif_wake_queue` — restart transmit

Synopsis

```
void netif_wake_queue (struct net_device * dev);
```

Arguments

dev

network device

Description

Allow upper layers to call the device hard_start_xmit routine. Used for flow control when transmit resources are available.

netif_stop_queue

LINUX

Name

`netif_stop_queue` — stop transmitted packets

Synopsis

```
void netif_stop_queue (struct net_device * dev);
```

Arguments

dev

network device

Description

Stop upper layers calling the device hard_start_xmit routine. Used for flow control when transmit resources are unavailable.

netif_queue_stopped

LINUX

Name

`netif_queue_stopped` — test if transmit queue is flowblocked

Synopsis

```
int netif_queue_stopped (const struct net_device * dev);
```

Arguments

dev

network device

Description

Test if transmit queue on device is currently unable to send.

netdev_txq_bql_enqueue_prefetchw

LINUX

Kernel Hackers ManualNovember 2026

Name

`netdev_txq_bql_enqueue_prefetchw` — prefetch bql data for write

Synopsis

```
void netdev_txq_bql_enqueue_prefetchw (struct netdev_queue *
dev_queue);
```

Arguments

dev_queue

pointer to transmit queue

Description

BQL enabled drivers might use this helper in their `ndo_start_xmit`, to give appropriate hint to the cpu.

netdev_txq_bql_complete_prefetchw

LINUX

Kernel Hackers ManualNovember 2026

Name

`netdev_txq_bql_complete_prefetchw` — prefetch bql data for write

Synopsis

```
void netdev_txq_bql_complete_prefetchw (struct netdev_queue *
dev_queue);
```

Arguments

dev_queue

pointer to transmit queue

Description

BQL enabled drivers might use this helper in their TX completion path, to give appropriate hint to the cpu.

netif_running

LINUX

Kernel Hackers ManualNovember 2026

Name

`netif_running` — test if up

Synopsis

```
int netif_running (const struct net_device * dev);
```

Arguments

dev

network device

Description

Test if the device has been brought up.

netif_start_subqueue

LINUX

Kernel Hackers ManualNovember 2026

Name

`netif_start_subqueue` — allow sending packets on subqueue

Synopsis

```
void netif_start_subqueue (struct net_device * dev, u16
queue_index);
```

Arguments

dev

network device

queue_index

sub queue index

Description

Start individual transmit queue of a device with multiple transmit queues.

netif_stop_subqueue

LINUX

Name

`netif_stop_subqueue` — stop sending packets on subqueue

Synopsis

```
void netif_stop_subqueue (struct net_device * dev, u16
queue_index);
```

Arguments

dev

network device

queue_index

sub queue index

Description

Stop individual transmit queue of a device with multiple transmit queues.

__netif_subqueue_stopped

LINUX

Name

`__netif_subqueue_stopped` — test status of subqueue

Synopsis

```
int __netif_subqueue_stopped (const struct net_device * dev,
u16 queue_index);
```

Arguments

dev

network device

queue_index

sub queue index

Description

Check individual transmit queue of a device with multiple transmit queues.

netif_wake_subqueue

LINUX

Kernel Hackers ManualNovember 2026

Name

`netif_wake_subqueue` — allow sending packets on subqueue

Synopsis

```
void netif_wake_subqueue (struct net_device * dev, u16
queue_index);
```

Arguments

dev

network device

queue_index

sub queue index

Description

Resume individual transmit queue of a device with multiple transmit queues.

netif_is_multiqueue

LINUX

Kernel Hackers ManualNovember 2026

Name

`netif_is_multiqueue` — test if device has multiple transmit queues

Synopsis

```
int netif_is_multiqueue (const struct net_device * dev);
```

Arguments

dev

network device

Description

Check if device has multiple transmit queues

dev_put

LINUX

Kernel Hackers ManualNovember 2026

Name

dev_put — release reference to device

Synopsis

```
void dev_put (struct net_device * dev);
```

Arguments

dev

network device

Description

Release reference to device to allow it to be freed.

dev_hold

LINUX

Kernel Hackers ManualNovember 2026

Name

dev_hold — get reference to device

Synopsis

```
void dev_hold (struct net_device * dev);
```

Arguments

dev

network device

Description

Hold reference to device to keep it from being freed.

netif_carrier_ok

LINUX

Name

`netif_carrier_ok` — test if carrier present

Synopsis

```
int netif_carrier_ok (const struct net_device * dev);
```

Arguments

dev

network device

Description

Check if carrier is present on device

netif_dormant_on

LINUX

Name

`netif_dormant_on` — mark device as dormant.

Synopsis

```
void netif_dormant_on (struct net_device * dev);
```

Arguments

dev

network device

Description

Mark device as dormant (as per RFC2863).

The dormant state indicates that the relevant interface is not actually in a condition to pass packets (i.e., it is not 'up') but is in a "pending" state, waiting for some external event. For "on- demand" interfaces, this new state identifies the situation where the interface is waiting for events to place it in the up state.

netif_dormant_off

LINUX

Kernel Hackers ManualNovember 2026

Name

`netif_dormant_off` — set device as not dormant.

Synopsis

```
void netif_dormant_off (struct net_device * dev);
```

Arguments

dev

network device

Description

Device is not in dormant state.

netif_dormant

LINUX

Kernel Hackers ManualNovember 2026

Name

`netif_dormant` — test if carrier present

Synopsis

```
int netif_dormant (const struct net_device * dev);
```

Arguments

dev

network device

Description

Check if carrier is present on device

netif_oper_up

LINUX

Kernel Hackers ManualNovember 2026

Name

`netif_oper_up` — test if device is operational

Synopsis

```
int netif_oper_up (const struct net_device * dev);
```

Arguments

dev

network device

Description

Check if carrier is operational

netif_device_present

LINUX

Name

`netif_device_present` — is device available or removed

Synopsis

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

Arguments

dev

network device

Description

Check if device has not been removed from system.

netif_tx_lock

LINUX

Name

`netif_tx_lock` — grab network device transmit lock

Synopsis

```
void netif_tx_lock (struct net_device * dev);
```

Arguments

dev

network device

Description

Get network device transmit lock

__dev_uc_sync

LINUX

Kernel Hackers ManualNovember 2026

Name

__dev_uc_sync — Synchonize device's unicast list

Synopsis

```
int __dev_uc_sync (struct net_device * dev, int (*sync)
(struct net_device *, const unsigned char *), int (*unsync)
(struct net_device *, const unsigned char *));
```

Arguments

dev

device to sync

sync

function to call if address should be added

unsync

function to call if address should be removed

Description

Add newly added addresses to the interface, and release addresses that have been deleted.

__dev_uc_unsync

LINUX

Kernel Hackers ManualNovember 2026

Name

__dev_uc_unsync — Remove synchonized addresses from device

Synopsis

```
void __dev_uc_unsync (struct net_device * dev, int (*unsync)
(struct net_device *, const unsigned char *));
```

Arguments

dev

device to sync

unsync

function to call if address should be removed

Description

Remove all addresses that were added to the device by `dev_uc_sync`.

__dev_mc_sync

LINUX

Kernel Hackers ManualNovember 2026

Name

`__dev_mc_sync` — Synchonize device's multicast list

Synopsis

```
int __dev_mc_sync (struct net_device * dev, int (*sync)
(struct net_device *, const unsigned char *), int (*unsync)
(struct net_device *, const unsigned char *));
```

Arguments

dev

device to sync

sync

function to call if address should be added

unsync

function to call if address should be removed

Description

Add newly added addresses to the interface, and release addresses that have been deleted.

__dev_mc_unsync

LINUX

Kernel Hackers ManualNovember 2026

Name

__dev_mc_unsync — Remove synchonized addresses from device

Synopsis

```
void __dev_mc_unsync (struct net_device * dev, int (*unsync)
(struct net_device *, const unsigned char *));
```

Arguments

dev

device to sync

unsync

function to call if address should be removed

Description

Remove all addresses that were added to the device by `dev_mc_sync`.

2.2. PHY Support

phy_print_status

LINUX

Kernel Hackers ManualNovember 2026

Name

`phy_print_status` — Convenience function to print out the current phy status

Synopsis

```
void phy_print_status (struct phy_device * phydev);
```

Arguments

phydev

the phy_device struct

phy_ethtool_sset

LINUX

Kernel Hackers ManualNovember 2026

Name

`phy_ethtool_sset` — generic ethtool sset function, handles all the details

Synopsis

```
int phy_ethtool_sset (struct phy_device * phydev, struct
ethtool_cmd * cmd);
```

Arguments

`phydev`

target phy_device struct

`cmd`

ethtool_cmd

A few notes about parameter checking

- We don't set port or transceiver, so we don't care what they were set to. - `phy_start_aneg` will make sure forced settings are sane, and choose the next best ones from the ones selected, so we don't care if ethtool tries to give us bad values.

phy_mii_ioctl

LINUX

Name

`phy_mii_ioctl` — generic PHY MII ioctl interface

Synopsis

```
int phy_mii_ioctl (struct phy_device * phydev, struct ifreq *
ifr, int cmd);
```

Arguments

`phydev`

the phy_device struct

`ifr`

struct ifreq for socket ioctl's

`cmd`

ioctl cmd to execute

Description

Note that this function is currently incompatible with the PHYCONTROL layer. It changes registers without regard to current state. Use at own risk.

phy_start_aneg

LINUX

Name

`phy_start_aneg` — start auto-negotiation for this PHY device

Synopsis

```
int phy_start_aneg (struct phy_device * phydev);
```

Arguments

`phydev`

the phy_device struct

Description

Sanitizes the settings (if we're not autonegotiating them), and then calls the driver's config_aneg function. If the PHYCONTROL Layer is operating, we change the state to reflect the beginning of Auto-negotiation or forcing.

phy_start_interrupts

LINUX

Name

`phy_start_interrupts` — request and enable interrupts for a PHY device

Synopsis

```
int phy_start_interrupts (struct phy_device * phydev);
```

Arguments

phydev

target phy_device struct

Description

Request the interrupt for the given PHY. If this fails, then we set irq to PHY_POLL. Otherwise, we enable the interrupts in the PHY. This should only be called with a valid IRQ number. Returns 0 on success or < 0 on error.

phy_stop_interrupts

LINUX

Kernel Hackers ManualNovember 2026

Name

phy_stop_interrupts — disable interrupts from a PHY device

Synopsis

```
int phy_stop_interrupts (struct phy_device * phydev);
```

Arguments

phydev

target phy_device struct

phy_stop

LINUX

Kernel Hackers ManualNovember 2026

Name

phy_stop — Bring down the PHY link, and stop checking the status

Synopsis

```
void phy_stop (struct phy_device * phydev);
```

Arguments

phydev

target phy_device struct

phy_start

LINUX

Name

`phy_start` — start or restart a PHY device

Synopsis

```
void phy_start (struct phy_device * phydev);
```

Arguments

`phydev`

target phy_device struct

Description

Indicates the attached device's readiness to handle PHY-related work. Used during startup to start the PHY, and after a call to `phy_stop` to resume operation. Also used to indicate the MDIO bus has cleared an error condition.

phy_clear_interrupt

LINUX

Name

`phy_clear_interrupt` — Ack the phy device's interrupt

Synopsis

```
int phy_clear_interrupt (struct phy_device * phydev);
```

Arguments

phydev

the phy_device struct

Description

If the *phydev* driver has an ack_interrupt function, call it to ack and clear the phy device's interrupt.

Returns 0 on success on < 0 on error.

phy_config_interrupt

LINUX

Kernel Hackers ManualNovember 2026

Name

phy_config_interrupt — configure the PHY device for the requested interrupts

Synopsis

```
int phy_config_interrupt (struct phy_device * phydev, u32
interrupts);
```

Arguments

phydev

the phy_device struct

interrupts

interrupt flags to configure for this *phydev*

Description

Returns 0 on success on < 0 on error.

phy_aneg_done

LINUX

Kernel Hackers ManualNovember 2026

Name

phy_aneg_done — return auto-negotiation status

Synopsis

```
int phy_aneg_done (struct phy_device * phydev);
```

Arguments

phydev

target phy_device struct

Description

Reads the status register and returns 0 either if auto-negotiation is incomplete, or if there was an error. Returns BMSR_ANEGCOMPLETE if auto-negotiation is done.

phy_find_setting

LINUX

Kernel Hackers ManualNovember 2026

Name

`phy_find_setting` — find a PHY settings array entry that matches speed & duplex

Synopsis

```
int phy_find_setting (int speed, int duplex);
```

Arguments

speed

speed to match

duplex

duplex to match

Description

Searches the settings array for the setting which matches the desired speed and duplex, and returns the index of that setting. Returns the index of the last setting if none of the others match.

phy_find_valid

LINUX

Kernel Hackers ManualNovember 2026

Name

`phy_find_valid` — find a PHY setting that matches the requested features mask

Synopsis

```
int phy_find_valid (int idx, u32 features);
```

Arguments

idx

The first index in settings[] to search

features

A mask of the valid settings

Description

Returns the index of the first valid setting less than or equal to the one pointed to by idx, as determined by the mask in features. Returns the index of the last setting if nothing else matches.

phy_sanitize_settings

LINUX

Kernel Hackers ManualNovember 2026

Name

`phy_sanitize_settings` — make sure the PHY is set to supported speed and duplex

Synopsis

```
void phy_sanitize_settings (struct phy_device * phydev);
```

Arguments

phydev

the target phy_device struct

Description

Make sure the PHY is set to supported speeds and duplexes. Drop down by one in this order: 1000/FULL, 1000/HALF, 100/FULL, 100/HALF, 10/FULL, 10/HALF.

phy_start_machine

LINUX

Name

phy_start_machine — start PHY state machine tracking

Synopsis

```
void phy_start_machine (struct phy_device * phydev, void
(*handler) (struct net_device *));
```

Arguments

phydev

the phy_device struct

handler

callback function for state change notifications

Description

The PHY infrastructure can run a state machine which tracks whether the PHY is starting up, negotiating, etc. This function starts the timer which tracks the state of the PHY. If you want to be notified when the state changes, pass in the callback *handler*, otherwise, pass NULL. If you want to maintain your own state machine, do not call this function.

phy_stop_machine

LINUX

Name

`phy_stop_machine` — stop the PHY state machine tracking

Synopsis

```
void phy_stop_machine (struct phy_device * phydev);
```

Arguments

`phydev`

target phy_device struct

Description

Stops the state machine timer, sets the state to UP (unless it wasn't up yet). This function must be called BEFORE phy_detach.

phy_force_reduction

LINUX

Name

`phy_force_reduction` — reduce PHY speed/duplex settings by one step

Synopsis

```
void phy_force_reduction (struct phy_device * phydev);
```

Arguments

phydev

target phy_device struct

Description

Reduces the speed/duplex settings by one notch, in this order-- 1000/FULL, 1000/HALF, 100/FULL, 100/HALF, 10/FULL, 10/HALF. The function bottoms out at 10/HALF.

phy_error

LINUX

Kernel Hackers ManualNovember 2026

Name

phy_error — enter HALTED state for this PHY device

Synopsis

```
void phy_error (struct phy_device * phydev);
```

Arguments

phydev

target phy_device struct

Description

Moves the PHY to the HALTED state in response to a read or write error, and tells the controller the link is down. Must not be called from interrupt context, or while the phydev->lock is held.

phy_interrupt

LINUX

Kernel Hackers ManualNovember 2026

Name

`phy_interrupt` — PHY interrupt handler

Synopsis

```
irqreturn_t phy_interrupt (int irq, void * phy_dat);
```

Arguments

irq

interrupt line

phy_dat

phy_device pointer

Description

When a PHY interrupt occurs, the handler disables interrupts, and schedules a work task to clear the interrupt.

phy_enable_interrupts

LINUX

Kernel Hackers ManualNovember 2026

Name

`phy_enable_interrupts` — Enable the interrupts from the PHY side

Synopsis

```
int phy_enable_interrupts (struct phy_device * phydev);
```

Arguments

`phydev`

target phy_device struct

phy_disable_interrupts

LINUX

Name

`phy_disable_interrupts` — Disable the PHY interrupts from the PHY side

Synopsis

```
int phy_disable_interrupts (struct phy_device * phydev);
```

Arguments

`phydev`

target phy_device struct

phy_change

LINUX

Name

`phy_change` — Scheduled by the phy_interrupt/timer to handle PHY changes

Synopsis

```
void phy_change (struct work_struct * work);
```

Arguments

work

work_struct that describes the work to be done

phy_state_machine

LINUX

Kernel Hackers ManualNovember 2026

Name

phy_state_machine — Handle the state machine

Synopsis

```
void phy_state_machine (struct work_struct * work);
```

Arguments

work

work_struct that describes the work to be done

get_phy_id

LINUX

Name

`get_phy_id` — reads the specified addr for its ID.

Synopsis

```
int get_phy_id (struct mii_bus * bus, int addr, u32 * phy_id);
```

Arguments

`bus`

the target MII bus

`addr`

PHY address on the MII bus

`phy_id`

where to store the ID retrieved.

Description

Reads the ID registers of the PHY at *`addr`* on the *`bus`*, stores it in *`phy_id`* and returns zero on success.

get_phy_device

LINUX

Name

`get_phy_device` — reads the specified PHY device and returns its *`phy_device`* struct

Synopsis

```
struct phy_device * get_phy_device (struct mii_bus * bus, int addr);
```

Arguments

`bus`

the target MII bus

`addr`

PHY address on the MII bus

Description

Reads the ID registers of the PHY at *`addr`* on the *`bus`*, then allocates and returns the phy_device to represent it.

phy_device_register

LINUX

Name

`phy_device_register` — Register the phy device on the MDIO bus

Synopsis

```
int phy_device_register (struct phy_device * phydev);
```

Arguments

`phydev`

phy_device structure to be added to the MDIO bus

phy_find_first

LINUX

Name

`phy_find_first` — finds the first PHY device on the bus

Synopsis

```
struct phy_device * phy_find_first (struct mii_bus * bus);
```

Arguments

bus

the target MII bus

phy_connect_direct

LINUX

Kernel Hackers ManualNovember 2026

Name

`phy_connect_direct` — connect an ethernet device to a specific phy_device

Synopsis

```
int phy_connect_direct (struct net_device * dev, struct
phy_device * phydev, void (*handler) (struct net_device *),
u32 flags, phy_interface_t interface);
```

Arguments

dev

the network device to connect

phydev

the pointer to the phy device

handler

callback function for state change notifications

flags

PHY device's dev_flags

interface

PHY device's interface

phy_connect

LINUX

Kernel Hackers ManualNovember 2026

Name

phy_connect — connect an ethernet device to a PHY device

Synopsis

```
struct phy_device * phy_connect (struct net_device * dev,
const char * bus_id, void (*handler) (struct net_device *),
u32 flags, phy_interface_t interface);
```

Arguments

dev

the network device to connect

bus_id

the id string of the PHY device to connect

handler

callback function for state change notifications

flags

PHY device's dev_flags

interface

PHY device's interface

Description

Convenience function for connecting ethernet devices to PHY devices. The default behavior is for the PHY infrastructure to handle everything, and only notify the connected driver when the link status changes. If you don't want, or can't use the provided functionality, you may choose to call only the subset of functions which provide the desired functionality.

phy_disconnect

LINUX

Kernel Hackers ManualNovember 2026

Name

`phy_disconnect` — disable interrupts, stop state machine, and detach a PHY device

Synopsis

```
void phy_disconnect (struct phy_device * phydev);
```

Arguments

phydev

target phy_device struct

phy_attach

LINUX

Kernel Hackers ManualNovember 2026

Name

phy_attach — attach a network device to a particular PHY device

Synopsis

```
struct phy_device * phy_attach (struct net_device * dev, const
char * bus_id, u32 flags, phy_interface_t interface);
```

Arguments

dev

network device to attach

bus_id

Bus ID of PHY device to attach

flags

PHY device's dev_flags

interface

PHY device's interface

Description

Same as `phy_attach_direct` except that a PHY bus_id string is passed instead of a pointer to a struct phy_device.

phy_detach

LINUX

Kernel Hackers ManualNovember 2026

Name

`phy_detach` — detach a PHY device from its network device

Synopsis

```
void phy_detach (struct phy_device * phydev);
```

Arguments

phydev

target phy_device struct

genphy_restart_aneg

LINUX

Kernel Hackers ManualNovember 2026

Name

`genphy_restart_aneg` — Enable and Restart Autonegotiation

Synopsis

```
int genphy_restart_aneg (struct phy_device * phydev);
```

Arguments

`phydev`

target phy_device struct

genphy_config_aneg

LINUX

Kernel Hackers ManualNovember 2026

Name

`genphy_config_aneg` — restart auto-negotiation or write BMCR

Synopsis

```
int genphy_config_aneg (struct phy_device * phydev);
```

Arguments

phydev

target phy_device struct

Description

If auto-negotiation is enabled, we configure the advertising, and then restart auto-negotiation. If it is not enabled, then we write the BMCR.

genphy_update_link

LINUX

Kernel Hackers ManualNovember 2026

Name

genphy_update_link — update link status in *phydev*

Synopsis

```
int genphy_update_link (struct phy_device * phydev);
```

Arguments

phydev

target phy_device struct

Description

Update the value in phydev->link to reflect the current link value. In order to do this, we need to read the status register twice, keeping the second value.

genphy_read_status

LINUX

Kernel Hackers ManualNovember 2026

Name

genphy_read_status — check the link status and update current link state

Synopsis

```
int genphy_read_status (struct phy_device * phydev);
```

Arguments

phydev

target phy_device struct

Description

Check the link, then figure out the current state by comparing what we advertise with what the link partner advertises. Start by checking the gigabit possibilities, then move on to 10/100.

phy_driver_register

LINUX

Kernel Hackers ManualNovember 2026

Name

`phy_driver_register` — register a phy_driver with the PHY layer

Synopsis

```
int phy_driver_register (struct phy_driver * new_driver);
```

Arguments

`new_driver`

new phy_driver to register

phy_prepare_link

LINUX

Name

`phy_prepare_link` — prepares the PHY layer to monitor link status

Synopsis

```
void phy_prepare_link (struct phy_device * phydev, void
(*handler) (struct net_device *));
```

Arguments

`phydev`

target phy_device struct

`handler`

callback function for link status change notifications

Description

Tells the PHY infrastructure to handle the gory details on monitoring link status (whether through polling or an interrupt), and to call back to the connected device driver when the link status changes. If you want to monitor your own link state, don't call this function.

phy_attach_direct

LINUX

Name

phy_attach_direct — attach a network device to a given PHY device pointer

Synopsis

```
int phy_attach_direct (struct net_device * dev, struct
phy_device * phydev, u32 flags, phy_interface_t interface);
```

Arguments

dev

network device to attach

phydev

Pointer to phy_device to attach

flags

PHY device's dev_flags

interface

PHY device's interface

Description

Called by drivers to attach to a particular PHY device. The phy_device is found, and properly hooked up to the phy_driver. If no driver is attached, then the genphy_driver is used. The phy_device is given a ptr to the attaching device, and given a callback for link status change. The phy_device is returned to the attaching driver.

genphy_config_advert

LINUX

Kernel Hackers ManualNovember 2026

Name

`genphy_config_advert` — sanitize and advertise auto-negotiation parameters

Synopsis

```
int genphy_config_advert (struct phy_device * phydev);
```

Arguments

`phydev`

target phy_device struct

Description

Writes MII_ADVERTISE with the appropriate values, after sanitizing the values to make sure we only advertise what is supported. Returns < 0 on error, 0 if the PHY's advertisement hasn't changed, and > 0 if it has changed.

genphy_setup_forced

LINUX

Name

`genphy_setup_forced` — configures/forces speed/duplex from *`phydev`*

Synopsis

```
int genphy_setup_forced (struct phy_device * phydev);
```

Arguments

`phydev`

target phy_device struct

Description

Configures MII_BMCR to force speed/duplex to the values in phydev. Assumes that the values are valid. Please see `phy_sanitize_settings`.

phy_probe

LINUX

Name

`phy_probe` — probe and init a PHY device

Synopsis

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

Arguments

dev

device to probe and init

Description

Take care of setting up the phy_device structure, set the state to READY (the driver's init function should set it to STARTING if needed).

mdiobus_alloc

LINUX

Kernel Hackers ManualNovember 2026

Name

`mdiobus_alloc` — allocate a mii_bus structure

Synopsis

```
struct mii_bus * mdiobus_alloc ( void);
```

Arguments

void

no arguments

Description

called by a bus driver to allocate an mii_bus structure to fill in.

mdiobus_register

LINUX

Kernel Hackers ManualNovember 2026

Name

mdiobus_register — bring up all the PHYs on a given bus and attach them to bus

Synopsis

```
int mdiobus_register (struct mii_bus * bus);
```

Arguments

bus

target mii_bus

Description

Called by a bus driver to bring up all the PHYs on a given bus, and attach them to the bus.

Returns 0 on success or < 0 on error.

mdiobus_free

LINUX

Kernel Hackers ManualNovember 2026

Name

`mdiobus_free` — free a struct mii_bus

Synopsis

```
void mdiobus_free (struct mii_bus * bus);
```

Arguments

bus

mii_bus to free

Description

This function releases the reference to the underlying device object in the mii_bus. If this is the last reference, the mii_bus will be freed.

mdiobus_read

LINUX

Kernel Hackers ManualNovember 2026

Name

`mdiobus_read` — Convenience function for reading a given MII mgmt register

Synopsis

```
int mdiobus_read (struct mii_bus * bus, int addr, u32 regnum);
```

Arguments

bus

the mii_bus struct

addr

the phy address

regnum

register number to read

NOTE

MUST NOT be called from interrupt context, because the bus read/write functions may wait for an interrupt to conclude the operation.

mdiobus_write

LINUX

Kernel Hackers ManualNovember 2026

Name

`mdiobus_write` — Convenience function for writing a given MII mgmt register

Synopsis

```
int mdiobus_write (struct mii_bus * bus, int addr, u32 regnum,
u16 val);
```

Arguments

bus

the mii_bus struct

addr

the phy address

regnum

register number to write

val

value to write to *regnum*

NOTE

MUST NOT be called from interrupt context, because the bus read/write functions may wait for an interrupt to conclude the operation.

mdiobus_release

LINUX

Kernel Hackers ManualNovember 2026

Name

`mdiobus_release` — mii_bus device release callback

Synopsis

```
void mdiobus_release (struct device * d);
```

Arguments

d

the target struct device that contains the mii_bus

Description

called when the last reference to an mii_bus is dropped, to free the underlying memory.

mdio_bus_match

LINUX

Name

`mdio_bus_match` — determine if given PHY driver supports the given PHY device

Synopsis

```
int mdio_bus_match (struct device * dev, struct device_driver
* drv);
```

Arguments

dev

target PHY device

drv

given PHY driver

Description

Given a PHY device, and a PHY driver, return 1 if the driver supports the device. Otherwise, return 0.