

# **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

<b>1. Linux Networking .....</b>	<b>1</b>
1.1. Networking Base Types .....	1
enum sock_type .....	1
struct socket .....	2
1.2. Socket Buffer Functions.....	3
struct sk_buff.....	4
skb_queue_empty .....	9
skb_get .....	10
skb_cloned .....	11
skb_header_cloned.....	11
skb_header_release .....	12
skb_shared.....	13
skb_share_check .....	14
skb_unshare.....	15
skb_peek .....	16
skb_peek_tail .....	17
skb_queue_len.....	17
__skb_queue_after .....	18
skb_put.....	19
skb_push .....	20
skb_pull.....	21
skb_headroom.....	22
skb_tailroom .....	23
skb_reserve .....	24
skb_trim .....	24
pskb_trim_unique .....	25
skb_orphan.....	26
__dev_alloc_skb .....	27
dev_alloc_skb .....	28
netdev_alloc_skb.....	29
skb_clone_writable .....	30
skb_cow .....	31
skb_cow_head.....	32
skb_padto .....	33
skb_linearize .....	34
skb_linearize_cow.....	35
skb_postpull_rcsum .....	35
pskb_trim_rcsum.....	36
skb_get_timestamp .....	37
skb_checksum_complete .....	38
struct sock_common .....	39

struct sock .....	41
sk_filter .....	47
sk_filter_release .....	48
sk_eat_skb.....	49
move_addr_to_kernel.....	50
move_addr_to_user.....	51
sockfd_lookup.....	52
sock_release .....	53
sock_register .....	54
sock_unregister .....	55
skb_over_panic .....	56
skb_under_panic .....	56
__alloc_skb .....	57
__netdev_alloc_skb.....	59
__kfree_skb.....	60
kfree_skb.....	60
skb_morph.....	61
skb_clone .....	62
skb_copy .....	63
pskb_copy .....	64
pskb_expand_head.....	65
skb_copy_expand.....	66
skb_pad .....	68
__pskb_pull_tail.....	68
skb_store_bits .....	69
skb_dequeue.....	70
skb_dequeue_tail.....	71
skb_queue_purge .....	72
skb_queue_head.....	73
skb_queue_tail .....	74
skb_unlink.....	75
skb_append .....	76
skb_insert.....	77
skb_split .....	78
skb_prepare_seq_read.....	79
skb_seq_read.....	80
skb_abort_seq_read.....	81
skb_find_text.....	82
skb_append_datato_frags.....	83
skb_pull_rcsum.....	84
skb_segment.....	85
skb_cow_data.....	86
skb_partial_csum_set.....	87

sk_alloc .....	88
sk_wait_data .....	89
__sk_mem_schedule .....	90
__sk_mem_reclaim .....	91
__skb_recv_datagram .....	92
skb_kill_datagram .....	93
skb_copy_datagram_iovec .....	94
skb_copy_and_csum_datagram_iovec .....	95
datagram_poll .....	97
sk_stream_write_space .....	98
sk_stream_wait_connect .....	98
sk_stream_wait_memory .....	99
1.3. Socket Filter .....	100
sk_run_filter .....	100
sk_chk_filter .....	101
1.4. Generic Network Statistics .....	102
struct gnet_stats_basic .....	102
struct gnet_stats_rate_est .....	103
struct gnet_stats_queue .....	104
struct gnet_estimator .....	105
gnet_stats_start_copy_compat .....	106
gnet_stats_start_copy .....	107
gnet_stats_copy_basic .....	108
gnet_stats_copy_rate_est .....	109
gnet_stats_copy_queue .....	110
gnet_stats_copy_app .....	111
gnet_stats_finish_copy .....	112
gen_new_estimator .....	113
gen_kill_estimator .....	114
gen_replace_estimator .....	115
1.5. SUN RPC subsystem .....	116
xdr_encode_opaque_fixed .....	117
xdr_encode_opaque .....	118
xdr_init_encode .....	119
xdr_reserve_space .....	120
xdr_write_pages .....	121
xdr_init_decode .....	122
xdr_inline_decode .....	122
xdr_read_pages .....	123
xdr_enter_page .....	124
svc_print_addr .....	125
svc_reserve .....	126
xpirt_register_transport .....	127

xprt_unregister_transport .....	128
xprt_reserve_xprt .....	129
xprt_release_xprt.....	130
xprt_release_xprt_cong.....	131
xprt_release_rqst_cong .....	131
xprt_adjust_cwnd.....	132
xprt_wake_pending_tasks.....	133
xprt_wait_for_buffer_space .....	134
xprt_write_space .....	134
xprt_set_retrans_timeout_def .....	135
xprt_disconnect_done .....	136
xprt_force_disconnect.....	137
xprt_lookup_rqst.....	137
xprt_update_rtt.....	138
xprt_complete_rqst .....	139
rpc_wake_up.....	140
rpc_wake_up_status.....	140
rpc_malloc.....	141
rpc_free .....	142
xdr_skb_read_bits.....	143
xdr_partial_copy_from_skb.....	144
csum_partial_copy_to_xdr.....	145
rpc_alloc_iostats .....	146
rpc_free_iostats.....	147
rpc_queue_upcall .....	147
rpc_mkpipe .....	148
rpc_unlink .....	150
rpcb_getport_sync.....	151
rpcb_getport_async.....	152
rpc_bind_new_program .....	153
rpc_run_task.....	154
rpc_call_sync .....	154
rpc_call_async.....	155
rpc_peeraddr .....	156
rpc_peeraddr2str .....	157
rpc_force_rebind.....	158
<b>2. Network device support.....</b>	<b>161</b>
2.1. Driver Support.....	161
dev_add_pack .....	161
__dev_remove_pack .....	161
dev_remove_pack .....	162
netdev_boot_setup_check .....	163
__dev_get_by_name .....	164

dev_get_by_name .....	165
__dev_get_by_index .....	166
dev_get_by_index .....	167
dev_getbyhwaddr .....	168
dev_get_by_flags .....	169
dev_valid_name .....	170
dev_alloc_name .....	171
netdev_features_change .....	172
netdev_state_change .....	173
dev_load .....	174
dev_open .....	175
dev_close .....	175
register_netdevice_notifier .....	176
unregister_netdevice_notifier .....	177
netif_device_detach .....	178
netif_device_attach .....	179
skb_gso_segment .....	180
dev_queue_xmit .....	180
netif_rx .....	182
netif_receive_skb .....	182
__napi_schedule .....	184
register_gifconf .....	184
netdev_set_master .....	185
dev_set_promiscuity .....	186
dev_set_allmulti .....	187
dev_unicast_delete .....	188
dev_unicast_add .....	189
dev_unicast_sync .....	190
dev_unicast_unsync .....	191
register_netdevice .....	192
register_netdev .....	193
alloc_netdev_mq .....	194
free_netdev .....	195
unregister_netdevice .....	196
unregister_netdev .....	197
netdev_compute_features .....	198
eth_header .....	199
eth_rebuild_header .....	200
eth_type_trans .....	201
eth_header_parse .....	202
eth_header_cache .....	203
eth_header_cache_update .....	203
ether_setup .....	204

alloc_etherdev_mq .....	205
netif_carrier_on .....	206
netif_carrier_off .....	207
is_zero_ether_addr .....	208
is_multicast_ether_addr .....	208
is_local_ether_addr .....	209
is_broadcast_ether_addr .....	210
is_valid_ether_addr .....	211
random_ether_addr .....	212
compare_ether_addr .....	212
napi_schedule_prep .....	213
napi_schedule .....	214
__napi_complete .....	215
napi_disable .....	216
napi_enable .....	217
napi_synchronize .....	217
netdev_priv .....	218
netif_napi_add .....	219
netif_start_queue .....	220
netif_wake_queue .....	221
netif_stop_queue .....	222
netif_queue_stopped .....	223
netif_running .....	224
netif_start_subqueue .....	224
netif_stop_subqueue .....	225
__netif_subqueue_stopped .....	226
netif_wake_subqueue .....	227
netif_is_multiqueue .....	228
dev_put .....	229
dev_hold .....	230
netif_carrier_ok .....	230
netif_dormant_on .....	231
netif_dormant_off .....	232
netif_dormant .....	233
netif_oper_up .....	234
netif_device_present .....	234
__netif_tx_lock .....	235
2.2. PHY Support .....	236
phy_print_status .....	236
phy_read .....	237
phy_write .....	238
phy_sanitize_settings .....	239
phy_ethtool_sset .....	240



phy_mii_ioctl .....	241
phy_start_aneg .....	241
phy_enable_interrupts .....	242
phy_disable_interrupts .....	243
phy_start_interrupts .....	244
phy_stop_interrupts .....	244
phy_stop .....	245
phy_start .....	246
phy_clear_interrupt .....	247
phy_config_interrupt .....	247
phy_aneg_done .....	248
phy_find_setting .....	249
phy_find_valid .....	250
phy_start_machine .....	251
phy_stop_machine .....	252
phy_force_reduction .....	253
phy_error .....	254
phy_interrupt .....	254
phy_change .....	255
phy_state_machine .....	256
phy_connect .....	257
phy_disconnect .....	258
phy_attach .....	259
phy_detach .....	260
genphy_config_advert .....	260
genphy_config_aneg .....	261
genphy_update_link .....	262
genphy_read_status .....	263
phy_driver_register .....	264
get_phy_device .....	264
phy_prepare_link .....	265
genphy_setup_forced .....	266
genphy_restart_aneg .....	267
phy_probe .....	268
mdiobus_register .....	269
mdio_bus_match .....	269
2.3. Synchronous PPP .....	270
sppp_close .....	271
sppp_open .....	271
sppp_reopen .....	272
sppp_do_ioctl .....	273
sppp_attach .....	274
sppp_detach .....	275



# Chapter 1. Linux Networking

## 1.1. Networking Base Types

### enum sock\_type

**LINUX**

Kernel Hackers Manual April 2008

#### 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 Manual April 2008

## Name

struct socket — general BSD socket

## Synopsis

```
struct socket {
    socket_state state;
    unsigned long flags;
    const struct proto_ops * ops;
    struct fasync_struct * fasync_list;
    struct file * file;
```

```
struct sock * sk;  
wait_queue_head_t wait;  
short type;  
};
```

## Members

state

socket state (SS\_CONNECTED, etc)

flags

socket flags (SOCK\_ASYNC\_NOSPACE, etc)

ops

protocol specific socket operations

fasync\_list

Asynchronous wake up list

file

File back pointer for gc

sk

internal networking protocol agnostic socket representation

wait

wait queue for several uses

type

socket type (SOCK\_STREAM, etc)

## 1.2. Socket Buffer Functions

### struct sk\_buff

**LINUX**

Kernel Hackers Manual April 2008

#### Name

struct sk\_buff — socket buffer

#### Synopsis

```
struct sk_buff {
    struct sk_buff * next;
    struct sk_buff * prev;
    struct sock * sk;
    ktime_t tstamp;
    struct net_device * dev;
    struct dst_entry * dst;
    struct sec_path * sp;
    char cb[48];
    unsigned int len;
    unsigned int data_len;
    __u16 mac_len;
    __u16 hdr_len;
    union {unnamed_union};
    __u32 priority;
    __u8 local_df:1;
    __u8 cloned:1;
    __u8 ip_summed:2;
    __u8 nohdr:1;
    __u8 nfctinfo:3;
    __u8 pkt_type:3;
    __u8 fclone:2;
    __u8 ipvs_property:1;
    __u8 peeked:1;
    __u8 nf_trace:1;
    __be16 protocol;
    void (* destructor) (struct sk_buff *skb);
#ifdef CONFIG_NF_CONNTRACK || defined(CONFIG_NF_CONNTRACK_MODULE)
    struct nf_conntrack * nfct;
#endif
};
```

```

    struct sk_buff * nfct_reasm;
#endif
#ifdef CONFIG_BRIDGE_NETFILTER
    struct nf_bridge_info * nf_bridge;
#endif
    int iif;
#ifdef CONFIG_NETDEVICES_MULTIQUEUE
    __u16 queue_mapping;
#endif
#ifdef CONFIG_NET_SCHED
    __u16 tc_index;
#endif
#ifdef CONFIG_NET_CLS_ACT
    __u16 tc_verd;
#endif
#endif
#ifdef CONFIG_XEN
#else
    __u8 proto_data_valid:1;
    __u8 proto_csum_blank:1;
#endif
#ifdef CONFIG_NET_DMA
    dma_cookie_t dma_cookie;
#endif
#ifdef CONFIG_NETWORK_SECMARK
    __u32 secmark;
#endif
    __u32 mark;
    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

## *Chapter 1. Linux Networking*

prev

Previous buffer in list

sk

Socket we are owned by

tstamp

Time we arrived

dev

Device we arrived on/are leaving by

dst

destination entry

sp

the security path, used for xfrm

cb[48]

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

len

Length of actual data

data\_len

Data length

mac\_len

Length of link layer header

hdr\_len

writable header length of cloned skb

{unnamed\_union}

anonymous

priority

Packet queueing priority

local\_df

allow local fragmentation



cloned

Head may be cloned (check refcnt to be sure)

ip\_summed

Driver fed us an IP checksum

nohdr

Payload reference only, must not modify header

nfctinfo

Relationship of this skb to the connection

pkt\_type

Packet class

fclone

skbuff clone status

ipvs\_property

skbuff is owned by ipvs

peeked

this packet has been seen already, so stats have been done for it, don't do them again

nf\_trace

netfilter packet trace flag

protocol

Packet protocol from driver

destructor

Destruct function

nfct

Associated connection, if any

nfct\_reasm

netfilter conntrack re-assembly pointer

## *Chapter 1. Linux Networking*

`nf_bridge`

Saved data about a bridged frame - see `br_netfilter.c`

`iif`

ifindex of device we arrived on

`queue_mapping`

Queue mapping for multiqueue devices

`tc_index`

Traffic control index

`tc_verd`

traffic control verdict

`proto_data_valid`

Protocol data validated since arriving at localhost

`proto_csum_blank`

Protocol csum must be added before leaving localhost

`dma_cookie`

a cookie to one of several possible DMA operations done by skb DMA functions

`secmark`

security marking

`mark`

Generic packet mark

`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\_queue\_empty

### LINUX

Kernel Hackers Manual April 2008

### Name

skb\_queue\_empty — check if a queue is empty

### Synopsis

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

## Arguments

*list*

queue head

## Description

Returns true if the queue is empty, false otherwise.

## skb\_get

### LINUX

Kernel Hackers Manual April 2008

## 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 Manual April 2008

## 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**

## 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

## 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

Kernel Hackers Manual April 2008

## 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

Kernel Hackers Manual April 2008

## 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 Manual April 2008

## 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 Manual April 2008

### Name

skb\_peek —

### Synopsis

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

### Arguments

*list\_*

list to peek at

### Description

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

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

## skb\_peek\_tail

### LINUX

Kernel Hackers Manual April 2008

### Name

`skb_peek_tail` —

### Synopsis

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

### Arguments

*list\_*

list to peek at

### Description

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

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

# skb\_queue\_len

## LINUX

Kernel Hackers Manual April 2008

### 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\_after

## LINUX

Kernel Hackers Manual April 2008

### 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 into 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\_put

**LINUX**

Kernel Hackers Manual April 2008

## 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 Manual April 2008

## 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 Manual April 2008

## Name

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

## Synopsis

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

## Arguments

*skb*

buffer to use

*len*

amount of data to remove

## Description

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

# skb\_headroom

## LINUX

Kernel Hackers Manual April 2008

## 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 Manual April 2008

## Name

`skb_tailroom` — bytes at buffer end

## Synopsis

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

## Arguments

*skb*

buffer to check

## Description

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

# skb\_reserve

## LINUX

Kernel Hackers Manual April 2008

## 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\_trim

## LINUX

Kernel Hackers Manual April 2008

### 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\_trim\_unique

## LINUX

## 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**

## 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

## 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.

## dev\_alloc\_skb

### LINUX

Kernel Hackers Manual April 2008

## 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.

## netdev\_alloc\_skb

### LINUX

Kernel Hackers Manual April 2008

## 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\_clone\_writable

### LINUX

Kernel Hackers Manual April 2008

## 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 Manual April 2008

## 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 Manual April 2008

## 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

Kernel Hackers Manual April 2008

## 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\_linearize

## LINUX

Kernel Hackers Manual April 2008

## Name

`skb_linearize` — convert paged skb to linear one

## Synopsis

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

## Arguments

*skb*

buffer to linearize

## Description

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

# skb\_linearize\_cow

## LINUX

Kernel Hackers Manual April 2008

## 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 Manual April 2008

### 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 Manual April 2008

### 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

## 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\_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.

## struct sock\_common

**LINUX**

## Name

`struct sock_common` — minimal network layer representation of sockets

## Synopsis

```
struct sock_common {
    unsigned short skc_family;
    volatile unsigned char skc_state;
    unsigned char skc_reuse;
    int skc_bound_dev_if;
    struct hlist_node skc_node;
    struct hlist_node skc_bind_node;
    atomic_t skc_refcnt;
    unsigned int skc_hash;
    struct proto * skc_prot;
    struct net * skc_net;
};
```

## Members

`skc_family`

network address family

`skc_state`

Connection state

`skc_reuse`

`SO_REUSEADDR` setting

`skc_bound_dev_if`

bound device index if != 0

`skc_node`

main hash linkage for various protocol lookup tables

`skc_bind_node`

bind hash linkage for various protocol lookup tables

`skc_refcnt`

reference count

`skc_hash`

hash value used with various protocol lookup tables

`skc_prot`

protocol handlers inside a network family

`skc_net`

reference to the network namespace of this socket

## Description

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

## struct sock

### LINUX

Kernel Hackers Manual April 2008

## Name

`struct sock` — network layer representation of sockets

## Synopsis

```
struct sock {
    struct sock_common __sk_common;
#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_node      __sk_common.skc_node
#define sk_bind_node __sk_common.skc_bind_node
```

```
#define sk_refcnt    __sk_common.skc_refcnt
#define sk_hash      __sk_common.skc_hash
#define sk_prot      __sk_common.skc_prot
#define sk_net       __sk_common.skc_net
    unsigned char sk_shutdown:2;
    unsigned char sk_no_check:2;
    unsigned char sk_userlocks:4;
    unsigned char sk_protocol;
    unsigned short sk_type;
    int sk_rcvbuf;
    socket_lock_t sk_lock;
    struct sk_backlog;
    wait_queue_head_t * sk_sleep;
    struct dst_entry * sk_dst_cache;
    struct xfrm_policy * sk_policy[2];
    rwlock_t sk_dst_lock;
    atomic_t sk_rmem_alloc;
    atomic_t sk_wmem_alloc;
    atomic_t sk_omem_alloc;
    int sk_sndbuf;
    struct sk_buff_head sk_receive_queue;
    struct sk_buff_head sk_write_queue;
    struct sk_buff_head sk_async_wait_queue;
    int sk_wmem_queued;
    int sk_forward_alloc;
    gfp_t sk_allocation;
    int sk_route_caps;
    int sk_gso_type;
    int sk_rcvlowat;
    unsigned long sk_flags;
    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;
    atomic_t sk_drops;
    unsigned short sk_ack_backlog;
    unsigned short sk_max_ack_backlog;
    __u32 sk_priority;
    struct ucred sk_peercred;
    long sk_rcvtimeo;
    long sk_sndtimeo;
    struct sk_filter * sk_filter;
    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;
void * sk_security;
__u32 sk_mark;
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_shutdown`

mask of `SEND_SHUTDOWN` and/or `RCV_SHUTDOWN`

`sk_no_check`

`SO_NO_CHECK` setting, whether or not checkup packets

`sk_userlocks`

`SO_SNDBUF` and `SO_RCVBUF` settings

`sk_protocol`

which protocol this socket belongs in this network family

`sk_type`

socket type (`SOCK_STREAM`, etc)

`sk_rcvbuf`

size of receive buffer in bytes

`sk_lock`

synchronizer

`sk_backlog`  
always used with the per-socket spinlock held

`sk_sleep`  
sock wait queue

`sk_dst_cache`  
destination cache

`sk_policy[2]`  
flow policy

`sk_dst_lock`  
destination cache lock

`sk_rmem_alloc`  
receive queue bytes committed

`sk_wmem_alloc`  
transmit queue bytes committed

`sk_omem_alloc`  
"o" is "option" or "other"

`sk_sndbuf`  
size of send buffer in bytes

`sk_receive_queue`  
incoming packets

`sk_write_queue`  
Packet sending queue

`sk_async_wait_queue`  
DMA copied packets

`sk_wmem_queued`  
persistent queue size

`sk_forward_alloc`  
space allocated forward

`sk_allocation`

allocation mode

`sk_route_caps`

route capabilities (e.g. `NETIF_F_TSO`)

`sk_gso_type`

GSO type (e.g. `SKB_GSO_TCPV4`)

`sk_rcvlowat`

`SO_RCVLOWAT` setting

`sk_flags`

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

`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_drops`

raw drops counter

`sk_ack_backlog`

current listen backlog

`sk_max_ack_backlog`

listen backlog set in `listen`

`sk_priority`

`SO_PRIORITY` setting

`sk_peercred`

`SO_PEERCRED` setting

`sk_rcvtimeo`

`SO_RCVTIMEO` setting

`sk_sndtimeo`

`SO_SNDTIMEO` setting

`sk_filter`

socket filtering instructions

`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_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`

## **sk\_filter**

**LINUX**

Kernel Hackers Manual April 2008

### **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\_filter\_release

### LINUX

Kernel Hackers Manual April 2008

## Name

sk\_filter\_release —

## 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\_eat\_skb

**LINUX**

Kernel Hackers Manual April 2008

## 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.

# move\_addr\_to\_kernel

## LINUX

Kernel Hackers Manual April 2008

## Name

`move_addr_to_kernel` — copy a socket address into kernel space

## Synopsis

```
int move_addr_to_kernel (void __user * uaddr, int ulen, void *  
kaddr);
```

## Arguments

*uaddr*

Address in user space

*ulen*

Length in user space

*kaddr*

Address in kernel space

## Description

The address is copied into kernel space. If the provided address is too long an error code of `-EINVAL` is returned. If the copy gives invalid addresses `-EFAULT` is returned. On a success 0 is returned.

# move\_addr\_to\_user

## LINUX

Kernel Hackers Manual April 2008

## Name

`move_addr_to_user` — copy an address to user space

## Synopsis

```
int move_addr_to_user (void * kaddr, int klen, void __user *  
uaddr, int __user * ulen);
```

## Arguments

*kaddr*

kernel space address

*klen*

length of address in kernel

*uaddr*

user space address

*ulen*

pointer to user length field

## Description

The value pointed to by *ulen* on entry is the buffer length available. This is overwritten with the buffer space used. -EINVAL is returned if an overlong buffer is specified or a negative buffer size. -EFAULT is returned if either the buffer or the length field are not accessible. After copying the data up to the limit the user specifies, the true length of the data is written over the length limit the user specified. Zero is returned for a success.

# sockfd\_lookup

## LINUX

Kernel Hackers Manual April 2008

## 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 Manual April 2008

## 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.

# sock\_register

## LINUX

Kernel Hackers Manual April 2008

## 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` corresponds to the socket system call protocol family.

# sock\_unregister

## LINUX

Kernel Hackers Manual April 2008

## 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.

# skb\_over\_panic

## LINUX

Kernel Hackers Manual April 2008

### Name

`skb_over_panic` — private function

### Synopsis

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

### Arguments

*skb*

buffer

*sz*

size

*here*

address

### Description

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

# skb\_under\_panic

## LINUX

Kernel Hackers Manual April 2008

### Name

skb\_under\_panic — private function

### Synopsis

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

### Arguments

*skb*

buffer

*sz*

size

*here*

address

### Description

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

# \_\_alloc\_skb

## LINUX

Kernel Hackers Manual April 2008

### Name

`__alloc_skb` — allocate a network buffer

### Synopsis

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

### Arguments

*size*

size to allocate

*gfp\_mask*

allocation mask

*fclone*

allocate from fclone cache instead of head cache and allocate a cloned (child)  
skb

*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 Manual April 2008

### 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.

## **\_\_kfree\_skb**

### **LINUX**

Kernel Hackers Manual April 2008

### **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

Kernel Hackers Manual April 2008

### Name

kfree\_skb — free an sk\_buff

### Synopsis

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

### Arguments

*skb*

buffer to free

### Description

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

# skb\_morph

## LINUX

Kernel Hackers Manual April 2008

### 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 Manual April 2008

## 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 Manual April 2008

### 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 Manual April 2008

## 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 Manual April 2008

## 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**

## 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 Manual April 2008

### 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.

# \_\_pskb\_pull\_tail

## LINUX

Kernel Hackers Manual April 2008

### 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 Manual April 2008

### 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 Manual April 2008

### 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

## 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

## 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 Manual April 2008

## 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 Manual April 2008

## 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 Manual April 2008

## 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 Manual April 2008

## 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 Manual April 2008

## 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 Manual April 2008

### Name

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

### Synopsis

```
void skb_split (struct sk_buff * skb, struct sk_buff * skbl,  
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

Kernel Hackers Manual April 2008

## 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 Manual April 2008

## 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 sequential 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**

## 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

## 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\_fragments

**LINUX**

## 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 Manual April 2008

### 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 Manual April 2008

## Name

`skb_segment` — Perform protocol segmentation on `skb`.

## Synopsis

```
struct sk_buff * skb_segment (struct sk_buff * skb, int
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**



## 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 Manual April 2008

### 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\_alloc

## LINUX

Kernel Hackers Manual April 2008

### 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

## 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**

## 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**

## Name

`__sk_mem_reclaim` — reclaim memory\_allocated

## Synopsis

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

## Arguments

*sk*  
socket

# `__skb_recv_datagram`

## LINUX

## 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

*skb*

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 Manual April 2008

### 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\_and\_csum\_datagram\_iovec

## LINUX

Kernel Hackers Manual April 2008

### Name

`skb_copy_and_csum_datagram_iovec` — Copy and checksum 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 Manual April 2008

### 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 Manual April 2008

## 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 Manual April 2008

### 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

## 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\_run\_filter

### LINUX

## Name

`sk_run_filter` — run a filter on a socket

## Synopsis

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

## Arguments

*skb*

buffer to run the filter on

*filter*

filter to apply

*flen*

length of filter

## Description

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

## sk\_chk\_filter

**LINUX**

Kernel Hackers Manual April 2008

## 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.

## 1.4. Generic Network Statistics

### struct gnet\_stats\_basic

**LINUX**



## 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

## 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 Manual April 2008

## 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 Manual April 2008

### 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 Manual April 2008

## 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 Manual April 2008

### 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 Manual April 2008

## Name

`gnet_stats_copy_basic` — copy basic statistics into statistic TLV

## Synopsis

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

## Arguments

*d*  
dumping handle

*b*  
basic statistics

## Description

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

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

# gnet\_stats\_copy\_rate\_est

## LINUX

Kernel Hackers Manual April 2008

## Name

`gnet_stats_copy_rate_est` — copy rate estimator statistics into statistics TLV

## Synopsis

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

## Arguments

*d*

dumping handle

*r*

rate estimator statistics

## Description

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

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

## `gnet_stats_copy_queue`

### LINUX

Kernel Hackers Manual April 2008

## 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 Manual April 2008

## 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 Manual April 2008

## 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 Manual April 2008

## Name

`gen_new_estimator` — create a new rate estimator

## Synopsis

```
int gen_new_estimator (struct gnet_stats_basic * bstats,  
struct gnet_stats_rate_est * rate_est, spinlock_t *  
stats_lock, struct 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 grabbed during this period.

Returns 0 on success or a negative error code.

## NOTE

Called under *rtnl\_mutex*

## gen\_kill\_estimator

**LINUX**

## Name

`gen_kill_estimator` — remove a rate estimator

## Synopsis

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

## Arguments

*bstats*

basic statistics

*rate\_est*

rate estimator statistics

## Description

Removes the rate estimator specified by *bstats* and *rate\_est* and deletes the timer.

## NOTE

Called under `rtnl_mutex`

## `gen_replace_estimator`

**LINUX**

## Name

`gen_replace_estimator` — replace rate estimator configuration

## Synopsis

```
int gen_replace_estimator (struct gnet_stats_basic * bstats,
struct gnet_stats_rate_est * rate_est, spinlock_t *
stats_lock, struct 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.

## 1.5. SUN RPC subsystem

### xdr\_encode\_opaque\_fixed

**LINUX**

Kernel Hackers Manual April 2008

#### 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 Manual April 2008

## 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\_init\_encode

## LINUX

Kernel Hackers Manual April 2008

## 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 Manual April 2008

### 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 Manual April 2008

## 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 Manual April 2008

### 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\_inline\_decode

**LINUX**

## Name

`xdr_inline_decode` — Retrieve non-page 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*.

# xprt\_register\_transport

## LINUX

Kernel Hackers Manual April 2008

## 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 Manual April 2008

### 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\_reserve\_xprt

**LINUX**

Kernel Hackers Manual April 2008

## Name

xprt\_reserve\_xprt — serialize write access to transports

## Synopsis

```
int xprt_reserve_xprt (struct rpc_task * task);
```

## Arguments

*task*

task that is requesting access to the transport

## Description

This prevents mixing the payload of separate requests, and prevents transport connects from colliding with writes. No congestion control is provided.

# xprt\_release\_xprt

## LINUX

Kernel Hackers Manual April 2008

## 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 Manual April 2008

### 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

## 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

## 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 Manual April 2008

## 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 Manual April 2008

## Name

`xprt_wait_for_buffer_space` — wait for transport output buffer to clear

## Synopsis

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

## Arguments

*task*

task to be put to sleep



# xprt\_write\_space

## LINUX

Kernel Hackers Manual April 2008

### 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 Manual April 2008

### 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

Kernel Hackers Manual April 2008

## 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\_force\_disconnect

**LINUX**

Kernel Hackers Manual April 2008

### Name

`xprt_force_disconnect` — force a transport to disconnect

### Synopsis

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

## Arguments

*xprt*

transport to 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\_update\_rtt

## LINUX

## Name

`xprt_update_rtt` — update an RPC client's RTT state after receiving a reply

## Synopsis

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

## Arguments

*task*

RPC request that recently completed

# xprt\_complete\_rqst

## LINUX

Kernel Hackers Manual April 2008

## 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 Manual April 2008

## 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 Manual April 2008

### 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` 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 Manual April 2008

### 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 Manual April 2008

### 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 Manual April 2008

## 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 Manual April 2008

## 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 Manual April 2008

## 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 Manual April 2008

## 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 Manual April 2008

### Name

`rpc_queue_upcall` —

### 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 Manual April 2008

### 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, 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`, 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 Manual April 2008

### 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\_sync

## LINUX

Kernel Hackers Manual April 2008

## Name

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

## Synopsis

```
int rpcb_getport_sync (struct sockaddr_in * sin, u32 prog, u32  
vers, int prot);
```

## Arguments

*sin*

address of remote peer

*prog*

RPC program number to bind

*vers*

RPC version number to bind

*prot*

transport protocol to use to make this request

## Description

Return value is the requested advertised port number, or a negative errno value.

Called from outside the RPC client in a synchronous task context. Uses default timeout parameters specified by underlying transport.

## XXX

Needs to support IPv6

# rpcb\_getport\_async

## LINUX

Kernel Hackers Manual April 2008

## 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 Manual April 2008

## 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 Manual April 2008

### 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

## Name

`rpc_call_sync` — Perform a synchronous RPC call

## Synopsis

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

## Arguments

*clnt*

pointer to RPC client

*msg*

RPC call parameters

*flags*

RPC call flags

## `rpc_call_async`

### LINUX

## Name

`rpc_call_async` — Perform an asynchronous RPC call

## Synopsis

```
int rpc_call_async (struct rpc_clnt * clnt, 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 Manual April 2008

## 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 Manual April 2008

## 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 Manual April 2008

## 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



# Chapter 2. Network device support

## 2.1. Driver Support

### dev\_add\_pack

**LINUX**

Kernel Hackers Manual April 2008

#### 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 Manual April 2008

### 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 Manual April 2008

## 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

### LINUX

Kernel Hackers Manual April 2008

### 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 Manual April 2008

## 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

## LINUX

Kernel Hackers Manual April 2008

## 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

## LINUX

Kernel Hackers Manual April 2008

## Name

`dev_getbyhwaddr` — find a device by its hardware address

## Synopsis

```
struct net_device * dev_getbyhwaddr (struct net * net,  
unsigned short type, 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 the rtnl semaphore. The returned device has not had its ref count increased and the caller must therefore be careful about locking

## BUGS

If the API was consistent this would be `__dev_get_by_hwaddr`

## dev\_get\_by\_flags

### LINUX

Kernel Hackers Manual April 2008

## Name

`dev_get_by_flags` — find any device with given flags

## Synopsis

```
struct net_device * dev_get_by_flags (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. 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\_valid\_name

**LINUX**

Kernel Hackers Manual April 2008

## 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 Manual April 2008

## 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 “%ld” 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 Manual April 2008

## 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 Manual April 2008

## 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

Kernel Hackers Manual April 2008

## 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

Kernel Hackers Manual April 2008

### 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

Kernel Hackers Manual April 2008

### Name

`dev_close` — shutdown an interface.

### Synopsis

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

### Arguments

*dev*

device to shutdown

### Description

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

# register\_netdevice\_notifier

## LINUX

## 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

## 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.

# netif\_device\_detach

## LINUX

Kernel Hackers Manual April 2008

## 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 Manual April 2008

## 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 Manual April 2008

### Name

`skb_gso_segment` — Perform segmentation on `skb`.

### Synopsis

```
struct sk_buff * skb_gso_segment (struct sk_buff * skb, int
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 Manual April 2008

### 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

# netif\_rx

## LINUX

Kernel Hackers Manual April 2008

### 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)



# netif\_receive\_skb

## LINUX

Kernel Hackers Manual April 2008

### 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 Manual April 2008

### **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

# register\_gifconf

## LINUX

Kernel Hackers Manual April 2008

### 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

## Name

`netdev_set_master` — set up master/slave pair

## Synopsis

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

## Arguments

*slave*

slave device

*master*

new master device

## Description

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

## `dev_set_promiscuity`

**LINUX**

## Name

`dev_set_promiscuity` — update promiscuity count on a device

## Synopsis

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

## Arguments

*dev*

device

*inc*

modifier

## Description

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

## `dev_set_allmulti`

**LINUX**

## Name

`dev_set_allmulti` — update allmulti count on a device

## Synopsis

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

## Arguments

*dev*

device

*inc*

modifier

## Description

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

## `dev_unicast_delete`

**LINUX**

## Name

`dev_unicast_delete` — Release secondary unicast address.

## Synopsis

```
int dev_unicast_delete (struct net_device * dev, void * addr,  
int alen);
```

## Arguments

*dev*

device

*addr*

address to delete

*alen*

length of *addr*

## Description

Release reference to a secondary unicast address and remove it from the device if the reference count drops to zero.

The caller must hold the `rtnl_mutex`.

## `dev_unicast_add`

**LINUX**

## Name

`dev_unicast_add` — add a secondary unicast address

## Synopsis

```
int dev_unicast_add (struct net_device * dev, void * addr, int  
alen);
```

## Arguments

*dev*

device

*addr*

address to delete

*alen*

length of *addr*

## Description

Add a secondary unicast address to the device or increase the reference count if it already exists.

The caller must hold the `rtnl_mutex`.

## `dev_unicast_sync`

**LINUX**



## Name

`dev_unicast_sync` — Synchronize device's unicast list to another device

## Synopsis

```
int dev_unicast_sync (struct net_device * to, struct
net_device * from);
```

## Arguments

*to*

destination device

*from*

source device

## Description

Add newly added addresses to the destination device and release addresses that have no users left. The source device must be locked by `netif_tx_lock_bh`.

This function is intended to be called from the `dev->set_rx_mode` function of layered software devices.

## `dev_unicast_unsync`

**LINUX**

## Name

`dev_unicast_unsync` — Remove synchronized addresses from the destination device

## Synopsis

```
void dev_unicast_unsync (struct net_device * to, struct  
net_device * from);
```

## Arguments

*to*

destination device

*from*

source device

## Description

Remove all addresses that were added to the destination device by `dev_unicast_sync`. This function is intended to be called from the `dev->stop` function of layered software devices.

# register\_netdevice

**LINUX**

## Name

`register_netdevice` — register a network device

## Synopsis

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

## Arguments

*dev*

device to register

## Description

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

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

## BUGS

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

# register\_netdev

## LINUX

Kernel Hackers Manual April 2008

### 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`.

# alloc\_netdev\_mq

## LINUX

## Name

`alloc_netdev_mq` — allocate network device

## Synopsis

```
struct net_device * alloc_netdev_mq (int sizeof_priv, const  
char * name, void (*setup) (struct net_device *), unsigned int  
queue_count);
```

## Arguments

*sizeof\_priv*

size of private data to allocate space for

*name*

device name format string

*setup*

callback to initialize device

*queue\_count*

the number of subqueues to allocate

## Description

Allocates a struct `net_device` with private data area for driver use and performs basic initialization. Also allocates subqueue structs for each queue on the device at the end of the netdevice.

# free\_netdev

## LINUX

Kernel Hackers Manual April 2008

### Name

`free_netdev` — free network device

### Synopsis

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

### Arguments

*dev*

device

### Description

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

# unregister\_netdevice

## LINUX

## Name

`unregister_netdevice` — remove device from the kernel

## Synopsis

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

## Arguments

*dev*

device

## Description

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

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

# 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`.

# netdev\_compute\_features

## LINUX

Kernel Hackers Manual April 2008

## Name

`netdev_compute_features` — compute conjunction of two feature sets

## Synopsis

```
int netdev_compute_features (unsigned long all, unsigned long  
one);
```



## Arguments

*all*

first feature set

*one*

second 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*. Returns the new feature set.

## eth\_header

### LINUX

Kernel Hackers Manual April 2008

### 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 ( $\leq \text{skb->len}$ )

## Description

Set the protocol type. For a packet of type `ETH_P_802_3` we put the length in here instead. It is up to the 802.2 layer to carry protocol information.

# eth\_rebuild\_header

**LINUX**

Kernel Hackers Manual April 2008

## 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

Kernel Hackers Manual April 2008

## 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

Kernel Hackers Manual April 2008

## 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 Manual April 2008

### 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

## 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.

## ether\_setup

**LINUX**

## 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\_mq

## LINUX

## Name

`alloc_etherdev_mq` — Allocates and sets up an Ethernet device

## Synopsis

```
struct net_device * alloc_etherdev_mq (int sizeof_priv,  
unsigned int queue_count);
```

## Arguments

*sizeof\_priv*

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

*queue\_count*

The number of 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 Manual April 2008

## 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 Manual April 2008

## 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.

# is\_zero\_ether\_addr

## LINUX

Kernel Hackers Manual April 2008

### 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

## 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

Kernel Hackers Manual April 2008

## 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\_valid\_ether\_addr

## LINUX

Kernel Hackers Manual April 2008

## 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.

# random\_ether\_addr

## LINUX

Kernel Hackers Manual April 2008

## Name

`random_ether_addr` — Generate software assigned random Ethernet address

## Synopsis

```
void random_ether_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.

# compare\_ether\_addr

## LINUX

Kernel Hackers Manual April 2008

### 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

# 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

## 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\_complete

### LINUX

Kernel Hackers Manual April 2008

## Name

\_\_napi\_complete — NAPI processing complete

## Synopsis

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

## Arguments

*n*

napi context

## Description

Mark NAPI processing as complete.

# napi\_disable

## LINUX

Kernel Hackers Manual April 2008

## 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

Kernel Hackers Manual April 2008

## 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

Kernel Hackers Manual April 2008

### 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

## 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\_napi\_add

## LINUX

## Name

`netif_napi_add` — initialize a napi context

## Synopsis

```
void netif_napi_add (struct net_device * dev, struct  
napi_struct * napi, int (*poll) (struct napi_struct *, int),  
int weight);
```

## Arguments

*dev*

network device

*napi*

napi context

*poll*

polling function

*weight*

default weight

## Description

`netif_napi_add` must be used to initialize a napi context prior to calling *any* of the other napi related functions.

## netif\_start\_queue

**LINUX**

## 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

## 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

Kernel Hackers Manual April 2008

## 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

Kernel Hackers Manual April 2008

## 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.

# netif\_running

## LINUX

Kernel Hackers ManualApril 2008

## 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 Manual April 2008

### 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 Manual April 2008

## 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 Manual April 2008

## 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 Always falls if  
NETDEVICE\_MULTIQUEUE is not configured

## dev\_put

### LINUX

Kernel Hackers Manual April 2008

## 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 Manual April 2008

## 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

Kernel Hackers Manual April 2008

### 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

Kernel Hackers Manual April 2008

### 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 Manual April 2008

## 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 Manual April 2008

## 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 Manual April 2008

### 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, int cpu);
```

## Arguments

*dev*

network device

*cpu*

cpu number of lock owner

## Description

Get network device transmit lock

## 2.2. PHY Support

### phy\_print\_status

**LINUX**

Kernel Hackers Manual April 2008

#### 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\_read

### LINUX

Kernel Hackers Manual April 2008

## Name

`phy_read` — Convenience function for reading a given PHY register

## Synopsis

```
int phy_read (struct phy_device * phydev, u16 regnum);
```

## Arguments

*phydev*

the `phy_device` struct

*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.

# phy\_write

## LINUX

Kernel Hackers Manual April 2008

## Name

`phy_write` — Convenience function for writing a given PHY register

## Synopsis

```
int phy_write (struct phy_device * phydev, u16 regnum, u16  
val);
```

## Arguments

*phydev*

the `phy_device` struct

*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.

# phy\_sanitize\_settings

## LINUX

Kernel Hackers Manual April 2008

## 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\_ethtool\_sset

## LINUX

Kernel Hackers Manual April 2008

## 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

Kernel Hackers Manual April 2008

### Name

`phy_mii_ioctl` — generic PHY MII ioctl interface

### Synopsis

```
int phy_mii_ioctl (struct phy_device * phydev, struct  
mii_ioctl_data * mii_data, int cmd);
```

### Arguments

*phydev*

the `phy_device` struct

*mii\_data*

MII ioctl data

*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

Kernel Hackers Manual April 2008

### 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\_enable\_interrupts

### LINUX

## 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\_start\_interrupts

### LINUX

Kernel Hackers Manual April 2008

## 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 Manual April 2008

### 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 Manual April 2008

### 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

Kernel Hackers Manual April 2008

## 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

Kernel Hackers Manual April 2008

## 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 or < 0 on error.

# phy\_config\_interrupt

## LINUX

Kernel Hackers Manual April 2008

### 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

## 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

## 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 Manual April 2008

## 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\_start\_machine

## LINUX

Kernel Hackers Manual April 2008

## 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

Kernel Hackers Manual April 2008

## 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

Kernel Hackers Manual April 2008

## 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 Manual April 2008

## 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 Manual April 2008

### 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\_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

## 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

## Description

Scheduled by the state\_queue workqueue each time phy\_timer is triggered.

# phy\_connect

## LINUX

Kernel Hackers Manual April 2008

## Name

phy\_connect — connect an ethernet device to a PHY device

## Synopsis

```
struct phy_device * phy_connect (struct net_device * dev,  
const char * phy_id, void (*handler) (struct net_device *),  
u32 flags, phy_interface_t interface);
```

## Arguments

*dev*

the network device to connect

*phy\_id*

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 Manual April 2008

## 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 Manual April 2008

## Name

`phy_attach` — attach a network device to a particular PHY device

## Synopsis

```
struct phy_device * phy_attach (struct net_device * dev, const  
char * phy_id, u32 flags, phy_interface_t interface);
```

## Arguments

*dev*

network device to attach

*phy\_id*

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.

## phy\_detach

### LINUX

Kernel Hackers Manual April 2008

## 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\_config\_advert

## LINUX

Kernel Hackers Manual April 2008

### 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.

# genphy\_config\_aneg

## LINUX

## 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

## 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 Manual April 2008

## 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 Manual April 2008

## 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

# get\_phy\_device

## LINUX

Kernel Hackers Manual April 2008

### 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\_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.

## 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`.

# genphy\_restart\_aneg

## LINUX

## Name

`genphy_restart_aneg` — Enable and Restart Autonegotiation

## Synopsis

```
int genphy_restart_aneg (struct phy_device * phydev);
```

## Arguments

*phydev*

target phy\_device struct

## phy\_probe

### LINUX

Kernel Hackers Manual April 2008

## 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\_register

## LINUX

Kernel Hackers Manual April 2008

## 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.

# mdio\_bus\_match

## LINUX

Kernel Hackers Manual April 2008

### 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.



## 2.3. Synchronous PPP

### sppp\_close

**LINUX**

Kernel Hackers Manual April 2008

#### Name

`sppp_close` — close down a synchronous PPP or Cisco HDLC link

#### Synopsis

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

#### Arguments

*dev*

The network device to drop the link of

#### Description

This drops the logical interface to the channel. It is not done politely as we assume we will also be dropping DTR. Any timeouts are killed.

### sppp\_open

**LINUX**

## Name

`sppp_open` — open a synchronous PPP or Cisco HDLC link

## Synopsis

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

## Arguments

*dev*

Network device to activate

## Description

Close down any existing synchronous session and commence from scratch. In the PPP case this means negotiating LCP/IPCPC and friends, while for Cisco HDLC we simply need to start sending keepalives

# sppp\_reopen

## LINUX

## Name

`sppp_reopen` — notify of physical link loss

## Synopsis

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

## Arguments

*dev*

Device that lost the link

## Description

This function informs the synchronous protocol code that the underlying link died (for example a carrier drop on X.21)

We increment the magic numbers to ensure that if the other end failed to notice we will correctly start a new session. It happens do to the nature of telco circuits is that you can lose carrier on one end only.

Having done this we go back to negotiating. This function may be called from an interrupt context.

## sppp\_do\_ioctl

### LINUX

Kernel Hackers Manual April 2008

## Name

sppp\_do\_ioctl — Ioctl handler for ppp/hdlc

## Synopsis

```
int sppp_do_ioctl (struct net_device * dev, struct ifreq *  
ifr, int cmd);
```

## Arguments

*dev*

Device subject to ioctl

*ifr*

Interface request block from the user

*cmd*

Command that is being issued

## Description

This function handles the ioctls that may be issued by the user to control the settings of a PPP/HDLC link. It does both busy and security checks. This function is intended to be wrapped by callers who wish to add additional ioctl calls of their own.

## sppp\_attach

### LINUX

Kernel Hackers Manual April 2008

## Name

`sppp_attach` — attach synchronous PPP/HDLC to a device

## Synopsis

```
void sppp_attach (struct ppp_device * pd);
```

## Arguments

*pd*

PPP device to initialise

## Description

This initialises the PPP/HDLC support on an interface. At the time of calling the `dev` element must point to the network device that this interface is attached to. The interface should not yet be registered.

## sppp\_detach

### LINUX

Kernel Hackers Manual April 2008

## Name

`sppp_detach` — release PPP resources from a device

## Synopsis

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

## **Arguments**

*dev*

Network device to release

## **Description**

Stop and free up any PPP/HDLC resources used by this interface. This must be called before the device is freed.