Linux Networking and Network Devices APIs

Linux Networking and Network Devices APIs

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

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

For more details see the file COPYING in the source distribution of Linux.

Table of Contents

1. Linux Networking 1
Networking Base Types 1
Socket Buffer Functions 3
Socket Filter 168
Generic Network Statistics 172
SUN RPC subsystem 188
WiMAX 263
2. Network device support 284
Driver Support 284
PHY Support 437

Chapter 1. Linux Networking

Networking Base Types

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.

Name

struct socket — general BSD socket

Synopsis

```
struct socket {
  socket_state state;
  short type;
  unsigned long flags;
  struct socket_wq __rcu * wq;
  struct file * file;
  struct sock * sk;
  const struct proto_ops * ops;
};
```

Members

state	socket state (`SS_CONNECTED`, etc)
type	socket type (`SOCK_STREAM`, etc)
flags	socket flags (`SOCK_ASYNC_NOSPACE`, etc)
wq	wait queue for several uses
file	File back pointer for gc
sk	internal networking protocol agnostic socket representation
ops	protocol specific socket operations

Socket Buffer Functions

Name

struct skb_shared_hwtstamps — hardware time stamps

Synopsis

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

Members

hwtstamp hardware time stamp transformed into duration since arbitrary point in time

Description

Software time stamps generated by `ktime_get_real` are stored in skb->tstamp.

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

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

Name

struct skb_mstamp — multi resolution time stamps

Synopsis

```
struct skb_mstamp {
  union {unnamed_union};
};
```

Members

{unnamed_union} anonymous

Name

skb_mstamp_get — get current timestamp

Synopsis

```
void skb_mstamp_get (struct skb_mstamp * cl);
```

Arguments

cl place to store timestamps

Name

skb_mstamp_us_delta — compute the difference in usec between two skb_mstamp

Synopsis

```
u32 skb_mstamp_us_delta (const struct skb_mstamp * t1, const struct
skb_mstamp * t0);
```

Arguments

t1 pointer to newest sample

t0 pointer to oldest sample

Name

struct sk_buff — socket buffer

Synopsis

```
struct sk_buff {
  union {unnamed_union};
  __u16 inner_transport_header;
  __u16 inner_network_header;
  __u16 inner_mac_header;
  __be16 protocol;
  __u16 transport_header;
  __u16 network_header;
  __u16 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

{unnamed_union}	anonymous
inner_transport_header	Inner transport layer header (encapsulation)
inner_network_header	Network layer header (encapsulation)
inner_mac_header	Link layer header (encapsulation)
protocol	Packet protocol from driver
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

Name

skb_dst — returns skb dst_entry

Synopsis

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

Arguments

skb buffer

Description

Returns skb dst_entry, regardless of reference taken or not.

Name

skb_dst_set — sets skb dst

Synopsis

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

Arguments

skb buffer

dst dst entry

Description

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

Name

skb_dst_set_noref — sets skb dst, hopefully, without taking reference

Synopsis

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

Arguments

skb buffer

dst dst entry

Description

Sets skb dst, assuming a reference was not taken on dst. If dst entry is cached, we do not take reference and dst_release will be avoided by refdst_drop. If dst entry is not cached, we take reference, so that last dst_release can destroy the dst immediately.

Name

skb_dst_is_noref — Test if skb dst isn't refcounted

Synopsis

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

Arguments

skb buffer

Name

skb_fclone_busy — check if fclone is busy

Synopsis

```
bool skb_fclone_busy (const struct sock * sk, const struct sk_buff *
skb);
```

Arguments

sk -- undescribed --

skb buffer

Description

Returns true is skb is a fast clone, and its clone is not freed. Some drivers call `skb_orphan` in their `ndo_start_xmit`, so we also check that this didnt happen.

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.

Name

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

Synopsis

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

Arguments

list queue head

skb buffer

Description

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

Name

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

Synopsis

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

Arguments

list queue head

skb buffer

Description

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

Name

skb_queue_next — return the next packet in the queue

Synopsis

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

Arguments

list queue head

skb current buffer

Description

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

Name

skb_queue_prev — return the prev packet in the queue

Synopsis

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

Arguments

list queue head

skb current buffer

Description

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

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.

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.

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.

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.

Note

Check if you can use `__skb_header_release` instead.

Name

__skb_header_release — release reference to header

Synopsis

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

Arguments

skb buffer to operate on

Description

Variant of skb_header_release assuming skb is private to caller. We can avoid one atomic operation.

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.

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.

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.

Name

skb_peek — peek at the head of an sk_buff_head

Synopsis

```
struct sk_buff * skb_peek (const 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.

Name

skb_peek_next — peek skb following the given one from a queue

Synopsis

```
struct sk_buff * skb_peek_next (struct sk_buff * skb, const struct
sk_buff_head * list_);
```

Arguments

skb skb to start from

*list_* list to peek at

Description

Returns NULL when the end of the list is met or a pointer to the next element. The reference count is not incremented and the reference is therefore volatile. Use with caution.

Name

skb_peek_tail — peek at the tail of an sk_buff_head

Synopsis

```
struct sk_buff * skb_peek_tail (const 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.

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.

Name

__skb_queue_head_init — initialize non-spinlock portions of sk_buff_head

Synopsis

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

Arguments

list queue to initialize

Description

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

Name

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

Synopsis

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

Arguments

list the new list to add

head the place to add it in the first list

Name

skb_queue_splice_init — join two skb lists and reinitialise the emptied list

Synopsis

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

Arguments

list the new list to add

head the place to add it in the first list

Description

The list at *list* is reinitialised

Name

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

Synopsis

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

Arguments

list the new list to add

head the place to add it in the first list

Name

skb_queue_splice_tail_init — join two skb lists and reinitialise the emptied list

Synopsis

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

Arguments

list the new list to add

head the place to add it in the first list

Description

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

Name

__skb_queue_after — queue a buffer at the list head

Synopsis

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

Arguments

list list to use

prev place after this buffer

newsk buffer to queue

Description

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

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

Name

__skb_fill_page_desc — initialise a paged fragment in an skb

Synopsis

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

Arguments

skb buffer containing fragment to be initialised

i paged fragment index to initialise

page the page to use for this fragment

off the offset to the data with *page*

size the length of the data

Description

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

Does not take any additional reference on the fragment.

Name

skb_fill_page_desc — initialise a paged fragment in an skb

Synopsis

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

Arguments

skb buffer containing fragment to be initialised

i paged fragment index to initialise

page the page to use for this fragment

off the offset to the data with *page*

size the length of the data

Description

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

Does not take any additional reference on the fragment.

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.

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

Name

skb_availroom — bytes at buffer end

Synopsis

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

Arguments

skb buffer to check

Description

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

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.

Name

skb_tailroom_reserve — adjust reserved_tailroom

Synopsis

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

Arguments

skb	buffer to alter
mtu	maximum amount of headlen permitted
needed_tailroom	minimum amount of reserved_tailroom

Description

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

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.

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.

Name

skb_orphan_frags — orphan the frags contained in a buffer

Synopsis

```
int skb_orphan_frags (struct sk_buff * skb, gfp_t gfp_mask);
```

Arguments

skb buffer to orphan frags from

gfp_mask allocation mask for replacement pages

Description

For each frag in the SKB which needs a destructor (i.e. has an owner) create a copy of that frag and release the original page by calling the destructor.

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.

Name

__dev_alloc_pages — allocate page for network Rx

Synopsis

```
struct page * __dev_alloc_pages (gfp_t gfp_mask, unsigned int order);
```

Arguments

gfp_mask allocation priority. Set __GFP_NOMEMALLOC if not for network Rx

order size of the allocation

Description

Allocate a new page.

NULL is returned if there is no free memory.

Name

__dev_alloc_page — allocate a page for network Rx

Synopsis

```
struct page * __dev_alloc_page (gfp_t gfp_mask);
```

Arguments

gfp_mask allocation priority. Set __GFP_NOMEMALLOC if not for network Rx

Description

Allocate a new page.

NULL is returned if there is no free memory.

Name

skb_propagate_pfmemalloc — Propagate pfmemalloc if skb is allocated after RX page

Synopsis

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

Arguments

page The page that was allocated from skb_alloc_page

skb The skb that may need pfmemalloc set

Name

skb_frag_page — retrieve the page referred to by a paged fragment

Synopsis

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

Arguments

frag the paged fragment

Description

Returns the struct page associated with *frag*.

Name

__skb_frag_ref — take an addition reference on a paged fragment.

Synopsis

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

Arguments

frag the paged fragment

Description

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

Name

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

Synopsis

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

Arguments

skb the buffer

f the fragment offset.

Description

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

Name

__skb_frag_unref — release a reference on a paged fragment.

Synopsis

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

Arguments

frag the paged fragment

Description

Releases a reference on the paged fragment *frag*.

Name

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

Synopsis

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

Arguments

skb the buffer

f the fragment offset

Description

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

Name

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

Synopsis

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

Arguments

frag the paged fragment buffer

Description

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

Name

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

Synopsis

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

Arguments

frag the paged fragment buffer

Description

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

Name

__skb_frag_set_page — sets the page contained in a paged fragment

Synopsis

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

Arguments

frag the paged fragment

page the page to set

Description

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

Name

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

Synopsis

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

Arguments

skb the buffer

f the fragment offset

page the page to set

Description

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

Name

skb_frag_dma_map — maps a paged fragment via the DMA API

Synopsis

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

Arguments

dev	the device to map the fragment to
frag	the paged fragment to map
offset	the offset within the fragment (starting at the fragment's own offset)
size	the number of bytes to map
dir	the direction of the mapping (PCI_DMA_*)

Description

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

Name

skb_clone_writable — is the header of a clone writable

Synopsis

```
int skb_clone_writable (const 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.

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.

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.

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.

Name

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

Synopsis

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

Arguments

skb buffer to pad

len minimal length

Description

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

Name

skb_linearize — convert paged skb to linear one

Synopsis

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

Arguments

skb buffer to linarize

Description

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

Name

skb_has_shared_frag — can any frag be overwritten

Synopsis

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

Arguments

skb buffer to test

Description

Return true if the skb has at least one frag that might be modified by an external entity (as in `vmsplice`/`sendfile`)

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.

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.

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.

Name

skb_needs_linearize — check if we need to linearize a given skb depending on the given device features.

Synopsis

```
bool skb_needs_linearize (struct sk_buff * skb, netdev_features_t features);
```

Arguments

skb socket buffer to check

features net device features

Returns true if either

1. skb has frag_list and the device doesn't support FRAGLIST, or 2. skb is fragmented and the device does not support SG.

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.

Name

skb_tx_timestamp — Driver hook for transmit timestamping

Synopsis

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

Arguments

skb A socket buffer.

Description

Ethernet MAC Drivers should call this function in their `hard_xmit` function immediately before giving the sk_buff to the MAC hardware.

Specifically, one should make absolutely sure that this function is called before TX completion of this packet can trigger. Otherwise the packet could potentially already be freed.

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.

Name

skb_checksum_none_assert — make sure skb ip_summed is CHECKSUM_NONE

Synopsis

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

Arguments

skb skb to check

Description

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

Name

skb_head_is_locked — Determine if the skb->head is locked down

Synopsis

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

Arguments

skb skb to check

Description

The head on skbs build around a head frag can be removed if they are not cloned. This function returns true if the skb head is locked down due to either being allocated via kmalloc, or by being a clone with multiple references to the head.

Name

skb_gso_network_seglen — Return length of individual segments of a gso packet

Synopsis

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

Arguments

skb GSO skb

Description

skb_gso_network_seglen is used to determine the real size of the individual segments, including Layer3 (IP, IPv6) and L4 headers (TCP/UDP).

The MAC/L2 header is not accounted for.

Name

struct sock_common — minimal network layer representation of sockets

Synopsis

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

Members

{unnamed_union}	anonymous
skc_tx_queue_mapping	tx queue number for this connection
skc_refcnt	reference count

Description

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

Name

struct sock — network layer representation of sockets

Synopsis

```
struct sock {
  struct sock_common __sk_common;
#define sk_node   __sk_common.skc_node
#define sk_nulls_node  __sk_common.skc_nulls_node
#define sk_refcnt  __sk_common.skc_refcnt
#define sk_tx_queue_mapping __sk_common.skc_tx_queue_mapping
#define sk_dontcopy_begin __sk_common.skc_dontcopy_begin
#define sk_dontcopy_end  __sk_common.skc_dontcopy_end
#define sk_hash   __sk_common.skc_hash
#define sk_portpair  __sk_common.skc_portpair
#define sk_num   __sk_common.skc_num
#define sk_dport  __sk_common.skc_dport
#define sk_addrpair  __sk_common.skc_addrpair
#define sk_daddr  __sk_common.skc_daddr
#define sk_rcv_saddr  __sk_common.skc_rcv_saddr
#define sk_family  __sk_common.skc_family
#define sk_state  __sk_common.skc_state
#define sk_reuse  __sk_common.skc_reuse
#define sk_reuseport  __sk_common.skc_reuseport
#define sk_ipv6only  __sk_common.skc_ipv6only
#define sk_bound_dev_if  __sk_common.skc_bound_dev_if
#define sk_bind_node  __sk_common.skc_bind_node
#define sk_prot   __sk_common.skc_prot
#define sk_net   __sk_common.skc_net
#define sk_v6_daddr  __sk_common.skc_v6_daddr
#define sk_v6_rcv_saddr __sk_common.skc_v6_rcv_saddr
#define sk_cookie  __sk_common.skc_cookie
  socket_lock_t sk_lock;
  struct sk_buff_head sk_receive_queue;
  struct sk_backlog;
#define sk_rmem_alloc sk_backlog.rmem_alloc
  int sk_forward_alloc;
#ifdef CONFIG_RPS
  __u32 sk_rxhash;
#endif
  u16 sk_incoming_cpu;
  __u32 sk_txhash;
#ifdef CONFIG_NET_RX_BUSY_POLL
  unsigned int sk_napi_id;
  unsigned int sk_ll_usec;
#endif
  atomic_t sk_drops;
  int sk_rcvbuf;
  struct sk_filter __rcu * sk_filter;
  struct socket_wq __rcu * sk_wq;
#ifdef CONFIG_XFRM
  struct xfrm_policy * sk_policy[2];
```

```
#endif
  unsigned long sk_flags;
  struct dst_entry * sk_rx_dst;
  struct dst_entry __rcu * sk_dst_cache;
  spinlock_t sk_dst_lock;
  atomic_t sk_wmem_alloc;
  atomic_t sk_omem_alloc;
  int sk_sndbuf;
  struct sk_buff_head sk_write_queue;
  unsigned int sk_shutdown:2;
  unsigned int sk_no_check_tx:1;
  unsigned int sk_no_check_rx:1;
  unsigned int sk_userlocks:4;
  unsigned int sk_protocol:8;
#define SK_PROTOCOL_MAX U8_MAX
  int sk_wmem_queued;
  gfp_t sk_allocation;
  u32 sk_pacing_rate;
  u32 sk_max_pacing_rate;
  netdev_features_t sk_route_caps;
  netdev_features_t sk_route_nocaps;
  int sk_gso_type;
  unsigned int sk_gso_max_size;
  u16 sk_gso_max_segs;
  int sk_rcvlowat;
  unsigned long sk_lingertime;
  struct sk_buff_head sk_error_queue;
  struct proto * sk_prot_creator;
  rwlock_t sk_callback_lock;
  int sk_err;
  int sk_err_soft;
  u32 sk_ack_backlog;
  u32 sk_max_ack_backlog;
  __u32 sk_priority;
#if IS_ENABLED(CONFIG_CGROUP_NET_PRIO)
  __u32 sk_cgrp_prioidx;
#endif
  struct pid * sk_peer_pid;
  const struct cred * sk_peer_cred;
  long sk_rcvtimeo;
  long sk_sndtimeo;
  void * sk_protinfo;
  struct timer_list sk_timer;
  ktime_t sk_stamp;
  u16 sk_tsflags;
  u32 sk_tskey;
  struct socket * sk_socket;
  void * sk_user_data;
  struct page_frag sk_frag;
  struct sk_buff * sk_send_head;
  __s32 sk_peek_off;
  int sk_write_pending;
#ifdef CONFIG_SECURITY
  void * sk_security;
```

```
#endif
  __u32 sk_mark;
  u32 sk_classid;
  struct cg_proto * sk_cgrp;
  void (* sk_state_change) (struct sock *sk);
  void (* sk_data_ready) (struct sock *sk);
  void (* sk_write_space) (struct sock *sk);
  void (* sk_error_report) (struct sock *sk);
  int (* sk_backlog_rcv) (struct sock *sk,struct sk_buff *skb);
  void (* sk_destruct) (struct sock *sk);
};
```

Members

__sk_common	shared layout with inet_timewait_sock
sk_lock	synchronizer
sk_receive_queue	incoming packets
sk_backlog	always used with the per-socket spinlock held
sk_forward_alloc	space allocated forward
sk_rxhash	flow hash received from netif layer
sk_incoming_cpu	record cpu processing incoming packets
sk_txhash	computed flow hash for use on transmit
sk_napi_id	id of the last napi context to receive data for sk
sk_ll_usec	usecs to busypoll when there is no data
sk_drops	raw/udp drops counter
sk_rcvbuf	size of receive buffer in bytes
sk_filter	socket filtering instructions
sk_wq	sock wait queue and async head
sk_policy[2]	flow policy
sk_flags	`SO_LINGER` (l_onoff), `SO_BROADCAST`, `SO_KEEPALIVE`, `SO_OOBINLINE` settings, `SO_TIMESTAMPING` settings
sk_rx_dst	receive input route used by early demux
sk_dst_cache	destination cache
sk_dst_lock	destination cache lock
sk_wmem_alloc	transmit queue bytes committed
sk_omem_alloc	"o“ is ”option“ or ”other"
sk_sndbuf	size of send buffer in bytes

sk_write_queue	Packet sending queue
sk_shutdown	mask of `SEND_SHUTDOWN` and/or `RCV_SHUTDOWN`
sk_no_check_tx	`SO_NO_CHECK` setting, set checksum in TX packets
sk_no_check_rx	allow zero checksum in RX packets
sk_userlocks	`SO_SNDBUF` and `SO_RCVBUF` settings
sk_protocol	which protocol this socket belongs in this network family
sk_wmem_queued	persistent queue size
sk_allocation	allocation mode
sk_pacing_rate	Pacing rate (if supported by transport/packet scheduler)
sk_max_pacing_rate	Maximum pacing rate (`SO_MAX_PACING_RATE`)
sk_route_caps	route capabilities (e.g. `NETIF_F_TSO`)
sk_route_nocaps	forbidden route capabilities (e.g NETIF_F_GSO_MASK)
sk_gso_type	GSO type (e.g. `SKB_GSO_TCPV4`)
sk_gso_max_size	Maximum GSO segment size to build
sk_gso_max_segs	Maximum number of GSO segments
sk_rcvlowat	`SO_RCVLOWAT` setting
sk_lingertime	`SO_LINGER` l_linger setting
sk_error_queue	rarely used
sk_prot_creator	sk_prot of original sock creator (see ipv6_setsockopt, IPV6_ADDRFORM for instance)
sk_callback_lock	used with the callbacks in the end of this struct
sk_err	last error
sk_err_soft	errors that don't cause failure but are the cause of a persistent failure not just 'timed out'
sk_ack_backlog	current listen backlog
sk_max_ack_backlog	listen backlog set in `listen`
sk_priority	`SO_PRIORITY` setting
sk_cgrp_prioidx	socket group's priority map index
sk_peer_pid	struct pid for this socket's peer
sk_peer_cred	`SO_PEERCRED` setting
sk_rcvtimeo	`SO_RCVTIMEO` setting

sk_sndtimeo	`SO_SNDTIMEO` setting
sk_protinfo	private area, net family specific, when not using slab
sk_timer	sock cleanup timer
sk_stamp	time stamp of last packet received
sk_tsflags	SO_TIMESTAMPING socket options
sk_tskey	counter to disambiguate concurrent tstamp requests
sk_socket	Identd and reporting IO signals
sk_user_data	RPC layer private data
sk_frag	cached page frag
sk_send_head	front of stuff to transmit
sk_peek_off	current peek_offset value
sk_write_pending	a write to stream socket waits to start
sk_security	used by security modules
sk_mark	generic packet mark
sk_classid	this socket's cgroup classid
sk_cgrp	this socket's cgroup-specific proto data
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

Name

sk_nulls_for_each_entry_offset — iterate over a list at a given struct offset

Synopsis

```
sk_nulls_for_each_entry_offset ( tpos, pos, head, offset);
```

Arguments

tpos	the type * to use as a loop cursor.
pos	the struct hlist_node to use as a loop cursor.
head	the head for your list.
offset	offset of hlist_node within the struct.

Name

unlock_sock_fast — complement of lock_sock_fast

Synopsis

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

Arguments

sk socket

slow slow mode

Description

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

Name

sk_wmem_alloc_get — returns write allocations

Synopsis

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

Arguments

sk socket

Description

Returns sk_wmem_alloc minus initial offset of one

Name

sk_rmem_alloc_get — returns read allocations

Synopsis

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

Arguments

sk socket

Description

Returns sk_rmem_alloc

Name

sk_has_allocations — check if allocations are outstanding

Synopsis

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

Arguments

sk socket

Description

Returns true if socket has write or read allocations

Name

wq_has_sleeper — check if there are any waiting processes

Synopsis

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

Arguments

wq struct socket_wq

Description

Returns true if socket_wq has waiting processes

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

Consider following tcp code paths

CPU1 CPU2

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

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

Name

sock_poll_wait — place memory barrier behind the poll_wait call.

Synopsis

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

Arguments

filp	file
wait_address	socket wait queue
p	poll_table

Description

See the comments in the wq_has_sleeper function.

Name

sk_page_frag — return an appropriate page_frag

Synopsis

```
struct page_frag * sk_page_frag (struct sock * sk);
```

Arguments

sk socket

Description

If socket allocation mode allows current thread to sleep, it means its safe to use the per task page_frag instead of the per socket one.

Name

sock_tx_timestamp — checks whether the outgoing packet is to be time stamped

Synopsis

```
void sock_tx_timestamp (const struct sock * sk, __u8 * tx_flags);
```

Arguments

sk socket sending this packet

tx_flags completed with instructions for time stamping

Note

callers should take care of initial *tx_flags value (usually 0)

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);
```

Arguments

sk socket to eat this skb from

skb socket buffer to eat

Description

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

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.

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.

Name

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

Synopsis

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

Arguments

sock The socket to receive the message from

msg Received message

vec Input s/g array for message data

num Size of input s/g array

size Number of bytes to read

flags Message flags (MSG_DONTWAIT, etc...)

Description

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

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

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.

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.

Name

__alloc_skb — allocate a network buffer

Synopsis

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

Arguments

size size to allocate

gfp_mask allocation mask

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

node numa node to allocate memory on

Description

Allocate a new sk_buff. The returned buffer has no headroom and a tail room of at least 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`.

Name

netdev_alloc_frag — allocate a page fragment

Synopsis

```
void * netdev_alloc_frag (unsigned int fragsz);
```

Arguments

fragsz fragment size

Description

Allocates a frag from a page for receive buffer. Uses GFP_ATOMIC allocations.

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 NET_SKB_PAD 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.

Name

__napi_alloc_skb — allocate skbuff for rx in a specific NAPI instance

Synopsis

```
struct sk_buff * __napi_alloc_skb (struct napi_struct * napi, unsigned
int length, gfp_t gfp_mask);
```

Arguments

napi	napi instance this buffer was allocated for
length	length to allocate
gfp_mask	get_free_pages mask, passed to alloc_skb and alloc_pages

Description

Allocate a new sk_buff for use in NAPI receive. This buffer will attempt to allocate the head from a special reserved region used only for NAPI Rx allocation. By doing this we can save several CPU cycles by avoiding having to disable and re-enable IRQs.

NULL is returned if there is no free memory.

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

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.

Name

skb_tx_error — report an sk_buff xmit error

Synopsis

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

Arguments

skb buffer that triggered an error

Description

Report xmit error if a device callback is tracking this skb. skb must be freed afterwards.

Name

consume_skb — free an skbuff

Synopsis

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

Arguments

skb buffer to free

Description

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

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.

Name

skb_copy_ubufs — copy userspace skb frags buffers to kernel

Synopsis

```
int skb_copy_ubufs (struct sk_buff * skb, gfp_t gfp_mask);
```

Arguments

skb the skb to modify

gfp_mask allocation priority

Description

This must be called on SKBTX_DEV_ZEROCOPY skb. It will copy all frags into kernel and drop the reference to userspace pages.

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

Returns 0 on success or a negative error code on failure to allocate kernel memory to copy to.

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`.

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.

Name

__pskb_copy_fclone — create copy of an sk_buff with private head.

Synopsis

```
struct sk_buff * __pskb_copy_fclone (struct sk_buff * skb, int headroom,
gfp_t gfp_mask, bool fclone);
```

Arguments

skb	buffer to copy
headroom	headroom of new skb
gfp_mask	allocation priority
fclone	if true allocate the copy of the skb from the fclone cache instead of the head cache; it is recommended to set this to true for the cases where the copy will likely be cloned

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.

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.

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.

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.

Name

pskb_put — add data to the tail of a potentially fragmented buffer

Synopsis

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

Arguments

skb start of the buffer to use

tail tail fragment of the buffer to use

len amount of data to add

Description

This function extends the used data area of the potentially fragmented buffer. *tail* must be the last fragment of *skb* -- or *skb* itself. If this would exceed the total buffer size the kernel will panic. A pointer to the first byte of the extra data is returned.

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.

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.

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.

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.

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.

Name

skb_copy_bits — copy bits from skb to kernel buffer

Synopsis

```
int skb_copy_bits (const struct sk_buff * skb, int offset, void * to,
int len);
```

Arguments

skb	source skb
offset	offset in source
to	destination buffer
len	number of bytes to copy

Description

Copy the specified number of bytes from the source skb to the destination buffer.

CAUTION ! : If its prototype is ever changed, check arch/{*}/net/{*}.S files, since it is called from BPF assembly code.

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.

Name

skb_zerocopy — Zero copy skb to skb

Synopsis

```
int skb_zerocopy (struct sk_buff * to, struct sk_buff * from, int len,
int hlen);
```

Arguments

to destination buffer

from source buffer

len number of bytes to copy from source buffer

hlen size of linear headroom in destination buffer

Description

Copies up to `len` bytes from `from` to `to` by creating references to the frags in the source buffer.

The `hlen` as calculated by `skb_zerocopy_headlen` specifies the headroom in the `to` buffer.

0

everything is OK -ENOMEM: couldn't orphan frags of *from* due to lack of memory -EFAULT: `skb_copy_bits` found some problem with skb geometry

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.

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.

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.

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.

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.

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.

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.

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.

Name

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

Synopsis

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

Arguments

skb the buffer to split

skb1 the buffer to receive the second part

len new length for skb

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.

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.

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.

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);
```

Arguments

skb	the buffer to look in
from	search offset
to	search limit
config	textsearch configuration

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.

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 appended 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

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.

Name

skb_segment — Perform protocol segmentation on skb.

Synopsis

```
struct sk_buff * skb_segment (struct sk_buff * head_skb,
netdev_features_t features);
```

Arguments

head_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).

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.

Name

skb_clone_sk — create clone of skb, and take reference to socket

Synopsis

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

Arguments

skb the skb to clone

Description

This function creates a clone of a buffer that holds a reference on sk_refcnt. Buffers created via this function are meant to be returned using sock_queue_err_skb, or free via kfree_skb.

When passing buffers allocated with this function to sock_queue_err_skb it is necessary to wrap the call with sock_hold/sock_put in order to prevent the socket from being released prior to being enqueued on the sk_error_queue.

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.

Name

skb_checksum_setup — set up partial checksum offset

Synopsis

```
int skb_checksum_setup (struct sk_buff * skb, bool recalculate);
```

Arguments

skb	the skb to set up
recalculate	if true the pseudo-header checksum will be recalculated

Name

skb_try_coalesce — try to merge skb to prior one

Synopsis

```
bool skb_try_coalesce (struct sk_buff * to, struct sk_buff * from, bool
* fragstolen, int * delta_truesize);
```

Arguments

to	prior buffer
from	buffer to add
fragstolen	pointer to boolean
delta_truesize	how much more was allocated than was requested

Name

skb_scrub_packet — scrub an skb

Synopsis

```
void skb_scrub_packet (struct sk_buff * skb, bool xnet);
```

Arguments

skb buffer to clean

xnet packet is crossing netns

Description

skb_scrub_packet can be used after encapsulating or decapsulting a packet into/from a tunnel. Some information have to be cleared during these operations. skb_scrub_packet can also be used to clean a skb before injecting it in another namespace (*xnet* == true). We have to clear all information in the skb that could impact namespace isolation.

Name

skb_gso_transport_seglen — Return length of individual segments of a gso packet

Synopsis

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

Arguments

skb GSO skb

Description

skb_gso_transport_seglen is used to determine the real size of the individual segments, including Layer4 headers (TCP/UDP).

The MAC/L2 or network (IP, IPv6) headers are not accounted for.

Name

alloc_skb_with_frags — allocate skb with page frags

Synopsis

```
struct sk_buff * alloc_skb_with_frags (unsigned long header_len,
unsigned long data_len, int max_page_order, int * errcode, gfp_t
gfp_mask);
```

Arguments

header_len	size of linear part
data_len	needed length in frags
max_page_order	max page order desired.
errcode	pointer to error code if any
gfp_mask	allocation mask

Description

This can be used to allocate a paged skb, given a maximal order for frags.

Name

sk_ns_capable — General socket capability test

Synopsis

```
bool sk_ns_capable (const struct sock * sk, struct user_namespace *
user_ns, int cap);
```

Arguments

sk	Socket to use a capability on or through
user_ns	The user namespace of the capability to use
cap	The capability to use

Description

Test to see if the opener of the socket had when the socket was created and the current process has the capability *cap* in the user namespace *user_ns*.

Name

sk_capable — Socket global capability test

Synopsis

```
bool sk_capable (const struct sock * sk, int cap);
```

Arguments

sk Socket to use a capability on or through

cap The global capability to use

Description

Test to see if the opener of the socket had when the socket was created and the current process has the capability *cap* in all user namespaces.

Name

sk_net_capable — Network namespace socket capability test

Synopsis

```
bool sk_net_capable (const struct sock * sk, int cap);
```

Arguments

sk Socket to use a capability on or through

cap The capability to use

Description

Test to see if the opener of the socket had when the socket was created and the current process has the capability *cap* over the network namespace the socket is a member of.

Name

sk_set_memalloc — sets SOCK_MEMALLOC

Synopsis

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

Arguments

sk socket to set it on

Description

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

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

Name

sk_clone_lock — clone a socket, and lock its clone

Synopsis

```
struct sock * sk_clone_lock (const struct sock * sk, const gfp_t priority);
```

Arguments

sk the socket to clone

priority for allocation (GFP_KERNEL, GFP_ATOMIC, etc)

Description

Caller must unlock socket even in error path (bh_unlock_sock(newsk))

Name

skb_page_frag_refill — check that a page_frag contains enough room

Synopsis

```
bool skb_page_frag_refill (unsigned int sz, struct page_frag * pfrag,
gfp_t gfp);
```

Arguments

sz minimum size of the fragment we want to get

pfrag pointer to page_frag

gfp priority for memory allocation

Note

While this allocator tries to use high order pages, there is no guarantee that allocations succeed. Therefore, *sz* MUST be less or equal than PAGE_SIZE.

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.

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.

Name

__sk_mem_reclaim — reclaim memory_allocated

Synopsis

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

Arguments

sk socket

Name

lock_sock_fast — fast version of lock_sock

Synopsis

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

Arguments

sk socket

Description

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

Name

__skb_recv_datagram — Receive a datagram skbuff

Synopsis

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

Arguments

sk socket

flags MSG_ flags

peeked returns non-zero if this packet has been seen before

off an offset in bytes to peek skb from. Returns an offset within an skb where data actually starts

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.

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.

Name

skb_copy_datagram_iter — Copy a datagram to an iovec iterator.

Synopsis

```
int skb_copy_datagram_iter (const struct sk_buff * skb, int offset,
struct iov_iter * to, int len);
```

Arguments

skb	buffer to copy
offset	offset in the buffer to start copying from
to	iovec iterator to copy to
len	amount of data to copy from buffer to iovec

Name

skb_copy_datagram_from_iter — Copy a datagram from an iov_iter.

Synopsis

```
int skb_copy_datagram_from_iter (struct sk_buff * skb, int offset,
struct iov_iter * from, int len);
```

Arguments

skb	buffer to copy
offset	offset in the buffer to start copying to
from	the copy source
len	amount of data to copy to buffer from iovec

Description

Returns 0 or -EFAULT.

Name

zerocopy_sg_from_iter — Build a zerocopy datagram from an iov_iter

Synopsis

```
int zerocopy_sg_from_iter (struct sk_buff * skb, struct iov_iter * from);
```

Arguments

skb buffer to copy

from the source to copy from

Description

The function will first copy up to headlen, and then pin the userspace pages and build frags through them.

Returns 0, -EFAULT or -EMSGSIZE.

Name

skb_copy_and_csum_datagram_msg — Copy and checksum skb to user iovec.

Synopsis

```
int skb_copy_and_csum_datagram_msg (struct sk_buff * skb, int hlen,
struct msghdr * msg);
```

Arguments

skb skbuff

hlen hardware length

msg destination

Description

Caller _must_ check that skb will fit to this iovec.

Returns

0 - success. -EINVAL - checksum failure. -EFAULT - fault during copy.

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.

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

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.

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

Socket Filter

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.

Name

bpf_check_classic — verify socket filter code

Synopsis

```
int bpf_check_classic (const struct sock_filter * filter, unsigned 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.

Name

bpf_prog_create — create an unattached filter

Synopsis

```
int bpf_prog_create (struct bpf_prog ** pfp, struct sock_fprog_kern *
fprog);
```

Arguments

pfp the unattached filter that is created

fprog the filter program

Description

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

Name

sk_attach_filter — attach a socket filter

Synopsis

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

Arguments

fprog the filter program

sk the socket to use

Description

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

Generic Network Statistics

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

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

Name

struct gnet_stats_rate_est64 — rate estimator

Synopsis

```
struct gnet_stats_rate_est64 {
  __u64 bps;
  __u64 pps;
};
```

Members

bps current byte rate

pps current packet rate

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

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

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.

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.

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_cpu __percpu * cpu, struct gnet_stats_basic_packed *
b);
```

Arguments

d dumping handle

cpu -- undescribed --

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.

Name

gnet_stats_copy_rate_est — copy rate estimator statistics into statistics TLV

Synopsis

```
int gnet_stats_copy_rate_est (struct gnet_dump * d, const struct
gnet_stats_basic_packed * b, struct gnet_stats_rate_est64 * r);
```

Arguments

d dumping handle

b basic statistics

r rate estimator statistics

Description

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

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

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
__percpu * cpu_q, struct gnet_stats_queue * q, __u32 qlen);
```

Arguments

d	dumping handle
cpu_q	per cpu queue statistics
q	queue statistics
qlen	queue length statistics

Description

Appends the queue statistics to the top level TLV created by `gnet_stats_start_copy`. Using per cpu queue statistics if they are available.

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

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 specific 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.

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.

Name

gen_new_estimator — create a new rate estimator

Synopsis

```
int gen_new_estimator (struct gnet_stats_basic_packed * bstats, struct gnet_stats_basic_cpu __percpu * cpu_bstats, struct gnet_stats_rate_est64 * rate_est, spinlock_t * stats_lock, struct nlattr * opt);
```

Arguments

bstats	basic statistics
cpu_bstats	-- undescribed --
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.

Name

gen_kill_estimator — remove a rate estimator

Synopsis

```
void gen_kill_estimator (struct gnet_stats_basic_packed * bstats, struct
gnet_stats_rate_est64 * rate_est);
```

Arguments

bstats basic statistics

rate_est rate estimator statistics

Description

Removes the rate estimator specified by bstats and rate_est.

Note

Caller should respect an RCU grace period before freeing stats_lock

Name

gen_replace_estimator — replace rate estimator configuration

Synopsis

```
int gen_replace_estimator (struct gnet_stats_basic_packed * bstats, struct gnet_stats_basic_cpu __percpu * cpu_bstats, struct gnet_stats_rate_est64 * rate_est, spinlock_t * stats_lock, struct nlattr * opt);
```

Arguments

bstats	basic statistics
cpu_bstats	-- undescribed --
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.

Name

gen_estimator_active — test if estimator is currently in use

Synopsis

```
bool gen_estimator_active (const struct gnet_stats_basic_packed *
bstats, const struct gnet_stats_rate_est64 * rate_est);
```

Arguments

bstats basic statistics

rate_est rate estimator statistics

Description

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

SUN RPC subsystem

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

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

Name

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

Synopsis

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

Arguments

buf XDR buffer where string resides

len length of string, in bytes

Name

_copy_from_pages —

Synopsis

```
void _copy_from_pages (char * p, struct page ** pages, size_t pgbase,
size_t len);
```

Arguments

p	pointer to destination
pages	array of pages
pgbase	offset of source data
len	length

Description

Copies data into an arbitrary memory location from an array of pages The copy is assumed to be non-overlapping.

Name

xdr_stream_pos — Return the current offset from the start of the xdr_stream

Synopsis

```
unsigned int xdr_stream_pos (const struct xdr_stream * xdr);
```

Arguments

xdr pointer to struct xdr_stream

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.

Name

xdr_commit_encode — Ensure all data is written to buffer

Synopsis

```
void xdr_commit_encode (struct xdr_stream * xdr);
```

Arguments

xdr pointer to xdr_stream

Description

We handle encoding across page boundaries by giving the caller a temporary location to write to, then later copying the data into place; xdr_commit_encode does that copying.

Normally the caller doesn't need to call this directly, as the following xdr_reserve_space will do it. But an explicit call may be required at the end of encoding, or any other time when the xdr_buf data might be read.

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.

Name

xdr_truncate_encode — truncate an encode buffer

Synopsis

```
void xdr_truncate_encode (struct xdr_stream * xdr, size_t len);
```

Arguments

xdr pointer to xdr_stream

len new length of buffer

Description

Truncates the xdr stream, so that xdr->buf->len == len, and xdr->p points at offset len from the start of the buffer, and head, tail, and page lengths are adjusted to correspond.

If this means moving xdr->p to a different buffer, we assume that that the end pointer should be set to the end of the current page, except in the case of the head buffer when we assume the head buffer's current length represents the end of the available buffer.

This is *not* safe to use on a buffer that already has inlined page cache pages (as in a zero-copy server read reply), except for the simple case of truncating from one position in the tail to another.

Name

xdr_restrict_buflen — decrease available buffer space

Synopsis

```
int xdr_restrict_buflen (struct xdr_stream * xdr, int newbuflen);
```

Arguments

xdr	pointer to xdr_stream
newbuflen	new maximum number of bytes available

Description

Adjust our idea of how much space is available in the buffer. If we've already used too much space in the buffer, returns -1. If the available space is already smaller than newbuflen, returns 0 and does nothing. Otherwise, adjusts xdr->buf->buflen to newbuflen and ensures xdr->end is set at most offset newbuflen from the start of the buffer.

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

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

Name

xdr_init_decode_pages — Initialize an xdr_stream for decoding data.

Synopsis

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

Arguments

xdr pointer to xdr_stream struct

buf pointer to XDR buffer from which to decode data

pages list of pages to decode into

len length in bytes of buffer in pages

Name

xdr_set_scratch_buffer — Attach a scratch buffer for decoding data.

Synopsis

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

Arguments

xdr	pointer to xdr_stream struct
buf	pointer to an empty buffer
buflen	size of 'buf'

Description

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

Name

xdr_inline_decode — Retrieve XDR data to decode

Synopsis

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

Arguments

xdr pointer to xdr_stream struct

nbytes number of bytes of data to decode

Description

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

Name

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

Synopsis

```
unsigned int 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[].

Returns the number of XDR encoded bytes now contained in the pages

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.

Name

xdr_buf_subsegment — set subbuf to a portion of buf

Synopsis

```
int xdr_buf_subsegment (struct xdr_buf * buf, struct xdr_buf * subbuf,
unsigned int base, unsigned int len);
```

Arguments

buf	an xdr buffer
subbuf	the result buffer
base	beginning of range in bytes
len	length of range in bytes

Description

sets *subbuf* to an xdr buffer representing the portion of *buf* of length *len* starting at offset *base*.

buf and *subbuf* may be pointers to the same struct xdr_buf.

Returns -1 if base of length are out of bounds.

Name

xdr_buf_trim — lop at most “len” bytes off the end of “buf”

Synopsis

```
void xdr_buf_trim (struct xdr_buf * buf, unsigned int len);
```

Arguments

buf buf to be trimmed

len number of bytes to reduce “buf” by

Description

Trim an xdr_buf by the given number of bytes by fixing up the lengths. Note that it's possible that we'll trim less than that amount if the xdr_buf is too small, or if (for instance) it's all in the head and the parser has already read too far into it.

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

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*.

Name

svc_find_xprt — find an RPC transport instance

Synopsis

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

Arguments

serv	pointer to svc_serv to search
xcl_name	C string containing transport's class name
net	owner net pointer
af	Address family of transport's local address
port	transport's IP port number

Description

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

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

Name

svc_xprt_names — format a buffer with a list of transport names

Synopsis

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

Arguments

serv	pointer to an RPC service
buf	pointer to a buffer to be filled in
buflen	length of buffer to be filled in

Description

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

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

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

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

Name

xprt_load_transport — load a transport implementation

Synopsis

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

Arguments

transport_name transport to load

0

transport successfully loaded -ENOENT: transport module not available

Name

xprt_reserve_xprt — serialize write access to transports

Synopsis

```
int xprt_reserve_xprt (struct rpc_xprt * xprt, struct rpc_task * task);
```

Arguments

xprt pointer to the target transport

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.

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.

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.

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.

Name

xprt_adjust_cwnd — adjust transport congestion window

Synopsis

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

Arguments

xprt pointer to xprt

task recently completed RPC request used to adjust window

result result code of completed RPC request

Description

The transport code maintains an estimate on the maximum number of out- standing RPC requests, using a smoothed version of the congestion avoidance implemented in 44BSD. This is basically the Van Jacobson

congestion algorithm

If a retransmit occurs, the congestion window is halved; otherwise, it is incremented by 1/cwnd when

- a reply is received and - a full number of requests are outstanding and - the congestion window hasn't been updated recently.

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

Name

xprt_wait_for_buffer_space — wait for transport output buffer to clear

Synopsis

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

Arguments

task task to be put to sleep

action function pointer to be executed after wait

Description

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

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.

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.

Name

xprt_set_retrans_timeout_rtt — set a request's retransmit timeout

Synopsis

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

Arguments

task task whose timeout is to be set

Description

Set a request's retransmit timeout using the RTT estimator.

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

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

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.

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

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

Name

rpc_malloc — allocate an RPC buffer

Synopsis

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

Arguments

task RPC task that will use this buffer

size requested byte size

Description

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

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

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

Name

rpc_free — free buffer allocated via rpc_malloc

Synopsis

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

Arguments

buffer buffer to free

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.

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

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

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

Name

rpc_free_iostats — release an rpc_iostats structure

Synopsis

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

Arguments

stats doomed rpc_iostats structure

Name

rpc_count_iostats_metrics — tally up per-task stats

Synopsis

```
void rpc_count_iostats_metrics (const struct rpc_task * task, struct
rpc_iostats * op_metrics);
```

Arguments

task	completed rpc_task
op_metrics	stat structure for OP that will accumulate stats from *task*

Name

rpc_count_iostats — tally up per-task stats

Synopsis

```
void rpc_count_iostats (const struct rpc_task * task, struct rpc_iostats
* stats);
```

Arguments

task completed rpc_task

stats array of stat structures

Description

Uses the statidx from *task*

Name

rpc_queue_upcall — queue an upcall message to userspace

Synopsis

```
int rpc_queue_upcall (struct rpc_pipe * pipe, struct rpc_pipe_msg *
msg);
```

Arguments

pipe 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.

Name

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

Synopsis

```
struct dentry * rpc_mkpipe_dentry (struct dentry * parent, const char
* name, void * private, struct rpc_pipe * pipe);
```

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

pipe rpc_pipe containing input parameters

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_inode(file))->private.

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.

Name

rpc_init_pipe_dir_head — initialise a struct rpc_pipe_dir_head

Synopsis

```
void rpc_init_pipe_dir_head (struct rpc_pipe_dir_head * pdh);
```

Arguments

pdh pointer to struct rpc_pipe_dir_head

Name

rpc_init_pipe_dir_object — initialise a struct rpc_pipe_dir_object

Synopsis

```
void rpc_init_pipe_dir_object (struct rpc_pipe_dir_object * pdo, const
struct rpc_pipe_dir_object_ops * pdo_ops, void * pdo_data);
```

Arguments

pdo pointer to struct rpc_pipe_dir_object

pdo_ops pointer to const struct rpc_pipe_dir_object_ops

pdo_data pointer to caller-defined data

Name

rpc_add_pipe_dir_object — associate a rpc_pipe_dir_object to a directory

Synopsis

```
int rpc_add_pipe_dir_object (struct net * net, struct rpc_pipe_dir_head
* pdh, struct rpc_pipe_dir_object * pdo);
```

Arguments

net pointer to struct net

pdh pointer to struct rpc_pipe_dir_head

pdo pointer to struct rpc_pipe_dir_object

Name

rpc_remove_pipe_dir_object — remove a rpc_pipe_dir_object from a directory

Synopsis

```
void rpc_remove_pipe_dir_object (struct net * net, struct
rpc_pipe_dir_head * pdh, struct rpc_pipe_dir_object * pdo);
```

Arguments

net pointer to struct net

pdh pointer to struct rpc_pipe_dir_head

pdo pointer to struct rpc_pipe_dir_object

Name

rpc_find_or_alloc_pipe_dir_object —

Synopsis

```
struct rpc_pipe_dir_object * rpc_find_or_alloc_pipe_dir_object (struct
net * net, struct rpc_pipe_dir_head * pdh, int (*match) (struct
rpc_pipe_dir_object *, void *), struct rpc_pipe_dir_object *(*alloc)
(void *), void * data);
```

Arguments

net pointer to struct net

pdh pointer to struct rpc_pipe_dir_head

match match struct rpc_pipe_dir_object to data

alloc allocate a new struct rpc_pipe_dir_object

data user defined data for `match` and `alloc`

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.

Name

rpc_create — create an RPC client and transport with one call

Synopsis

```
struct rpc_clnt * rpc_create (struct rpc_create_args * args);
```

Arguments

args rpc_clnt create argument structure

Description

Creates and initializes an RPC transport and an RPC client.

It can ping the server in order to determine if it is up, and to see if it supports this program and version. RPC_CLNT_CREATE_NOPING disables this behavior so asynchronous tasks can also use rpc_create.

Name

rpc_clone_client — Clone an RPC client structure

Synopsis

```
struct rpc_clnt * rpc_clone_client (struct rpc_clnt * clnt);
```

Arguments

clnt RPC client whose parameters are copied

Description

Returns a fresh RPC client or an ERR_PTR.

Name

rpc_clone_client_set_auth — Clone an RPC client structure and set its auth

Synopsis

```
struct rpc_clnt * rpc_clone_client_set_auth (struct rpc_clnt * clnt,
rpc_authflavor_t flavor);
```

Arguments

clnt RPC client whose parameters are copied

flavor security flavor for new client

Description

Returns a fresh RPC client or an ERR_PTR.

Name

rpc_switch_client_transport —

Synopsis

```
int rpc_switch_client_transport (struct rpc_clnt * clnt, struct
xprt_create * args, const struct rpc_timeout * timeout);
```

Arguments

clnt pointer to a struct rpc_clnt

args pointer to the new transport arguments

timeout pointer to the new timeout parameters

Description

This function allows the caller to switch the RPC transport for the rpc_clnt structure 'clnt' to allow it to connect to a mirrored NFS server, for instance. It assumes that the caller has ensured that there are no active RPC tasks by using some form of locking.

Returns zero if “clnt” is now using the new xprt. Otherwise a negative errno is returned, and “clnt” continues to use the old xprt.

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, const
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.

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

Name

rpc_call_sync — Perform a synchronous RPC call

Synopsis

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

Arguments

clnt pointer to RPC client

msg RPC call parameters

flags RPC call flags

Name

rpc_call_async — Perform an asynchronous RPC call

Synopsis

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

Arguments

clnt	pointer to RPC client
msg	RPC call parameters
flags	RPC call flags
tk_ops	RPC call ops
data	user call data

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.

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

NB

the lifetime of the memory referenced by the returned pointer is the same as the rpc_xprt itself. As long as the caller uses this pointer, it must hold the RCU read lock.

Name

rpc_localaddr — discover local endpoint address for an RPC client

Synopsis

```
int rpc_localaddr (struct rpc_clnt * clnt, struct sockaddr * buf, size_t
buflen);
```

Arguments

clnt RPC client structure

buf target buffer

buflen size of target buffer, in bytes

Description

Returns zero and fills in “buf” and “buflen” if successful; otherwise, a negative errno is returned.

This works even if the underlying transport is not currently connected, or if the upper layer never previously provided a source address.

The result of this function call is transient

multiple calls in succession may give different results, depending on how local networking configuration changes over time.

Name

rpc_protocol — Get transport protocol number for an RPC client

Synopsis

```
int rpc_protocol (struct rpc_clnt * clnt);
```

Arguments

clnt RPC client to query

Name

rpc_net_ns — Get the network namespace for this RPC client

Synopsis

```
struct net * rpc_net_ns (struct rpc_clnt * clnt);
```

Arguments

clnt RPC client to query

Name

rpc_max_payload — Get maximum payload size for a transport, in bytes

Synopsis

```
size_t rpc_max_payload (struct rpc_clnt * clnt);
```

Arguments

clnt RPC client to query

Description

For stream transports, this is one RPC record fragment (see RFC 1831), as we don't support multi-record requests yet. For datagram transports, this is the size of an IP packet minus the IP, UDP, and RPC header sizes.

Name

rpc_get_timeout — Get timeout for transport in units of HZ

Synopsis

```
unsigned long rpc_get_timeout (struct rpc_clnt * clnt);
```

Arguments

clnt RPC client to query

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

WiMAX

Name

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

Synopsis

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

Arguments

wimax_dev	WiMAX device descriptor
pipe_name	"named pipe" the message will be sent to
msg	pointer to the message data to send
size	size of the message to send (in bytes), including the header.
gfp_flags	flags for memory allocation.

Returns

`0` if ok, negative errno code on error

Description

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

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

IMPORTANT

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

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

Name

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

Synopsis

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

Arguments

msg Pointer to a message created with `wimax_msg_alloc`

size Pointer to where to store the message's size

Description

Returns the pointer to the message data.

Name

wimax_msg_data — Return a pointer to a message's payload

Synopsis

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

Arguments

msg Pointer to a message created with `wimax_msg_alloc`

Name

wimax_msg_len — Return a message's payload length

Synopsis

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

Arguments

msg Pointer to a message created with `wimax_msg_alloc`

Name

wimax_msg_send — Send a pre-allocated message to user space

Synopsis

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

Arguments

wimax_dev WiMAX device descriptor

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

Returns

0 if ok, < 0 errno code on error

Description

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

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

IMPORTANT

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

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

Name

wimax_msg — Send a message to user space

Synopsis

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

Arguments

wimax_dev	WiMAX device descriptor (properly referenced)
pipe_name	"named pipe" the message will be sent to
buf	pointer to the message to send.
size	size of the buffer pointed to by *buf* (in bytes).
gfp_flags	flags for memory allocation.

Returns

`0` if ok, negative errno code on error.

Description

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

NOTES

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

Name

wimax_reset — Reset a WiMAX device

Synopsis

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

Arguments

wimax_dev WiMAX device descriptor

Returns

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

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

`-EINVAL` if the device is not even registered.

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

Description

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

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

Name

wimax_report_rfkill_hw — Reports changes in the hardware RF switch

Synopsis

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

Arguments

wimax_dev WiMAX device descriptor

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

Description

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

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

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

Name

wimax_report_rfkill_sw — Reports changes in the software RF switch

Synopsis

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

Arguments

wimax_dev WiMAX device descriptor

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

Description

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

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

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

Name

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

Synopsis

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

Arguments

wimax_dev WiMAX device descriptor

state New RF state.

Returns

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

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

Description

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

NOTE

This call will block until the operation is complete.

Name

wimax_state_change — Set the current state of a WiMAX device

Synopsis

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

Arguments

wimax_dev WiMAX device descriptor (properly referenced)

new_state New state to switch to

Description

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

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

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

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

NOTE

wimax_dev must be locked

Name

wimax_state_get — Return the current state of a WiMAX device

Synopsis

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

Arguments

wimax_dev WiMAX device descriptor

Returns

Current state of the device according to its driver.

Name

wimax_dev_init — initialize a newly allocated instance

Synopsis

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

Arguments

wimax_dev WiMAX device descriptor to initialize.

Description

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

Name

wimax_dev_add — Register a new WiMAX device

Synopsis

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

Arguments

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

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

Description

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

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

Name

wimax_dev_rm — Unregister an existing WiMAX device

Synopsis

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

Arguments

wimax_dev WiMAX device descriptor

Description

Unregisters a WiMAX device previously registered for use with `wimax_add_rm`.

IMPORTANT! Must call before calling `unregister_netdev`.

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

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

Name

struct wimax_dev — Generic WiMAX device

Synopsis

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

Members

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

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

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

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

state — [private] Current state of the WiMAX device.

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

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

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

	complete. In case of having had to resort to bus/cold reset implying a device disconnection, the call is allowed to return immediately.
rfkill	[private] integration into the RF-Kill infrastructure.
rf_hw	[private] State of the hardware radio switch (OFF/ON)
rf_sw	[private] State of the software radio switch (OFF/ON)
name[32]	[fill] A way to identify this device. We need to register a name with many subsystems (rfkill, workqueue creation, etc). We can't use the network device name as that might change and in some instances we don't know it yet (until we don't call `register_netdev`). So we generate an unique one using the driver name and device bus id, place it here and use it across the board. Recommended naming: DRIVERNAME-BUSNAME:BUSID (dev->bus->name, dev->bus_id).
debugfs_dentry	[private] Used to hook up a debugfs entry. This shows up in the debugfs root as wimax\:DEVICENAME.

NOTE

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

Description

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

Usage

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

2. `memset` it to zero

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

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

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

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

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

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

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

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

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

CONCURRENCY

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

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

REFERENCE COUNTING

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

RFKILL

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

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

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

Name

enum wimax_st — The different states of a WiMAX device

Synopsis

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

Constants

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

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

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

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

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

WIMAX_ST_READY
: The device is fully up and radio is on.

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

WIMAX_ST_CONNECTING
: The device is connecting to a network. This state exists because in some devices, the connect process can include a number of

	negotiations between user space, kernel space and the device. User space needs to know what the device is doing. If the connect sequence in a device is atomic and fast, the device can transition directly to CONNECTED
WIMAX_ST_CONNECTED	The device is connected to a network.
__WIMAX_ST_INVALID	This is an invalid state used to mark the maximum numeric value of states.

Description

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

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

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

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

Chapter 2. Network device support

Driver Support

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).

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.

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.

Name

dev_add_offload — register offload handlers

Synopsis

```
void dev_add_offload (struct packet_offload * po);
```

Arguments

po protocol offload declaration

Description

Add protocol offload handlers to the networking stack. The passed proto_offload 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 offload handlers (until the next received packet).

Name

dev_remove_offload — remove packet offload handler

Synopsis

```
void dev_remove_offload (struct packet_offload * po);
```

Arguments

po packet offload declaration

Description

Remove a packet offload handler that was previously added to the kernel offload handlers by `dev_add_offload`. The passed offload_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.

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.

Name

dev_get_iflink — get 'iflink' value of a interface

Synopsis

```
int dev_get_iflink (const struct net_device * dev);
```

Arguments

dev targeted interface

Description

Indicates the ifindex the interface is linked to. Physical interfaces have the same 'ifindex' and 'iflink' values.

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.

Name

dev_get_by_name_rcu — find a device by its name

Synopsis

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

Arguments

net the applicable net namespace

name name to find

Description

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

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.

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*.

Name

dev_get_by_index_rcu — find a device by its ifindex

Synopsis

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

Arguments

net the applicable net namespace

ifindex index of device

Description

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

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.

Name

dev_getbyhwaddr_rcu — find a device by its hardware address

Synopsis

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

Arguments

net the applicable net namespace

type media type of device

ha hardware address

Description

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

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. Must be called inside `rtnl_lock`, and result refcount is unchanged.

Name

dev_valid_name — check if name is okay for network device

Synopsis

```
bool 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.

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 "lt%d" 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.

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.

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.

Name

netdev_notify_peers — notify network peers about existence of *dev*

Synopsis

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

Arguments

dev network device

Description

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

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.

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.

Name

dev_disable_lro — disable Large Receive Offload on a device

Synopsis

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

Arguments

dev device

Description

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

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.

Name

unregister_netdevice_notifier — unregister a network notifier block

Synopsis

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

Arguments

nb notifier

Description

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

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

Name

call_netdevice_notifiers — call all network notifier blocks

Synopsis

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

Arguments

val value passed unmodified to notifier function

dev net_device pointer passed unmodified to notifier function

Description

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

Name

dev_forward_skb — loopback an skb to another netif

Synopsis

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

Arguments

dev destination network device

skb buffer to forward

return values

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

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

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

Name

netif_set_real_num_rx_queues — set actual number of RX queues used

Synopsis

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

Arguments

dev Network device

rxq Actual number of RX queues

Description

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

Name

netif_get_num_default_rss_queues — default number of RSS queues

Synopsis

```
int netif_get_num_default_rss_queues ( void);
```

Arguments

void no arguments

Description

This routine should set an upper limit on the number of RSS queues used by default by multiqueue devices.

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.

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.

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.

Name

skb_mac_gso_segment — mac layer segmentation handler.

Synopsis

```
struct sk_buff * skb_mac_gso_segment (struct sk_buff * skb,
netdev_features_t features);
```

Arguments

skb buffer to segment

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

Name

__skb_gso_segment — Perform segmentation on skb.

Synopsis

```
struct sk_buff * __skb_gso_segment (struct sk_buff * skb,
netdev_features_t features, bool tx_path);
```

Arguments

skb	buffer to segment
features	features for the output path (see dev->features)
tx_path	whether it is called in TX path

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.

Segmentation preserves SKB_SGO_CB_OFFSET bytes of previous skb cb.

Name

dev_loopback_xmit — loop back *skb*

Synopsis

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

Arguments

sk -- undescribed --

skb buffer to transmit

Name

rps_may_expire_flow — check whether an RFS hardware filter may be removed

Synopsis

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

Arguments

dev	Device on which the filter was set
rxq_index	RX queue index
flow_id	Flow ID passed to `ndo_rx_flow_steer`
filter_id	Filter ID returned by `ndo_rx_flow_steer`

Description

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

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)

Name

netdev_rx_handler_register — register receive handler

Synopsis

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

Arguments

dev	device to register a handler for
rx_handler	receive handler to register
rx_handler_data	data pointer that is used by rx handler

Description

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

The caller must hold the rtnl_mutex.

For a general description of rx_handler, see enum rx_handler_result.

Name

netdev_rx_handler_unregister — unregister receive handler

Synopsis

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

Arguments

dev device to unregister a handler from

Description

Unregister a receive handler from a device.

The caller must hold the rtnl_mutex.

Name

netif_receive_skb_sk — process receive buffer from network

Synopsis

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

Arguments

sk -- undescribed --

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

Name

__napi_schedule — schedule for receive

Synopsis

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

Arguments

n entry to schedule

Description

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

Name

__napi_schedule_irqoff — schedule for receive

Synopsis

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

Arguments

n entry to schedule

Description

Variant of `__napi_schedule` assuming hard irqs are masked

Name

netdev_has_upper_dev — Check if device is linked to an upper device

Synopsis

```
bool netdev_has_upper_dev (struct net_device * dev, struct net_device
* upper_dev);
```

Arguments

dev device

upper_dev upper device to check

Description

Find out if a device is linked to specified upper device and return true in case it is. Note that this checks only immediate upper device, not through a complete stack of devices. The caller must hold the RTNL lock.

Name

netdev_master_upper_dev_get — Get master upper device

Synopsis

```
struct net_device * netdev_master_upper_dev_get (struct net_device *
dev);
```

Arguments

dev device

Description

Find a master upper device and return pointer to it or NULL in case it's not there. The caller must hold the RTNL lock.

Name

netdev_upper_get_next_dev_rcu — Get the next dev from upper list

Synopsis

```
struct net_device * netdev_upper_get_next_dev_rcu (struct net_device *
dev, struct list_head ** iter);
```

Arguments

dev device

iter list_head ** of the current position

Description

Gets the next device from the dev's upper list, starting from iter position. The caller must hold RCU read lock.

Name

netdev_all_upper_get_next_dev_rcu — Get the next dev from upper list

Synopsis

```
struct net_device * netdev_all_upper_get_next_dev_rcu (struct net_device * dev, struct list_head ** iter);
```

Arguments

dev device

iter list_head ** of the current position

Description

Gets the next device from the dev's upper list, starting from iter position. The caller must hold RCU read lock.

Name

netdev_lower_get_next_private — Get the next ->private from the lower neighbour list

Synopsis

```
void * netdev_lower_get_next_private (struct net_device * dev, struct
list_head ** iter);
```

Arguments

dev device

iter list_head ** of the current position

Description

Gets the next netdev_adjacent->private from the dev's lower neighbour list, starting from iter position. The caller must hold either hold the RTNL lock or its own locking that guarantees that the neighbour lower list will remain unchainged.

Name

netdev_lower_get_next_private_rcu — Get the next ->private from the lower neighbour list, RCU variant

Synopsis

```
void * netdev_lower_get_next_private_rcu (struct net_device * dev,
struct list_head ** iter);
```

Arguments

dev device

iter list_head ** of the current position

Description

Gets the next netdev_adjacent->private from the dev's lower neighbour list, starting from iter position. The caller must hold RCU read lock.

Name

netdev_lower_get_next — Get the next device from the lower neighbour list

Synopsis

```
void * netdev_lower_get_next (struct net_device * dev, struct list_head
** iter);
```

Arguments

dev device

iter list_head ** of the current position

Description

Gets the next netdev_adjacent from the dev's lower neighbour list, starting from iter position. The caller must hold RTNL lock or its own locking that guarantees that the neighbour lower list will remain unchainged.

Name

netdev_lower_get_first_private_rcu — Get the first ->private from the lower neighbour list, RCU variant

Synopsis

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

Arguments

dev device

Description

Gets the first netdev_adjacent->private from the dev's lower neighbour list. The caller must hold RCU read lock.

Name

netdev_master_upper_dev_get_rcu — Get master upper device

Synopsis

```
struct net_device * netdev_master_upper_dev_get_rcu (struct net_device
* dev);
```

Arguments

dev device

Description

Find a master upper device and return pointer to it or NULL in case it's not there. The caller must hold the RCU read lock.

Name

netdev_upper_dev_link — Add a link to the upper device

Synopsis

```
int netdev_upper_dev_link (struct net_device * dev, struct net_device
* upper_dev);
```

Arguments

dev device

upper_dev new upper device

Description

Adds a link to device which is upper to this one. The caller must hold the RTNL lock. On a failure a negative errno code is returned. On success the reference counts are adjusted and the function returns zero.

Name

netdev_master_upper_dev_link — Add a master link to the upper device

Synopsis

```
int netdev_master_upper_dev_link (struct net_device * dev, struct net_device * upper_dev);
```

Arguments

dev device

upper_dev new upper device

Description

Adds a link to device which is upper to this one. In this case, only one master upper device can be linked, although other non-master devices might be linked as well. The caller must hold the RTNL lock. On a failure a negative errno code is returned. On success the reference counts are adjusted and the function returns zero.

Name

netdev_upper_dev_unlink — Removes a link to upper device

Synopsis

```
void netdev_upper_dev_unlink (struct net_device * dev, struct net_device
* upper_dev);
```

Arguments

dev device

upper_dev new upper device

Description

Removes a link to device which is upper to this one. The caller must hold the RTNL lock.

Name

netdev_bonding_info_change — Dispatch event about slave change

Synopsis

```
void netdev_bonding_info_change (struct net_device * dev, struct
netdev_bonding_info * bonding_info);
```

Arguments

dev	device
bonding_info	info to dispatch

Description

Send NETDEV_BONDING_INFO to netdev notifiers with info. The caller must hold the RTNL lock.

Name

dev_set_promiscuity — update promiscuity count on a device

Synopsis

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

Arguments

dev device

inc modifier

Description

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

Name

dev_set_allmulti — update allmulti count on a device

Synopsis

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

Arguments

dev device

inc modifier

Description

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

Name

dev_get_flags — get flags reported to userspace

Synopsis

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

Arguments

dev device

Description

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

Name

dev_change_flags — change device settings

Synopsis

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

Arguments

dev device

flags device state flags

Description

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

Name

dev_set_mtu — Change maximum transfer unit

Synopsis

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

Arguments

dev device

new_mtu new transfer unit

Description

Change the maximum transfer size of the network device.

Name

dev_set_group — Change group this device belongs to

Synopsis

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

Arguments

dev device

new_group group this device should belong to

Name

dev_set_mac_address — Change Media Access Control Address

Synopsis

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

Arguments

dev device

sa new address

Description

Change the hardware (MAC) address of the device

Name

dev_change_carrier — Change device carrier

Synopsis

```
int dev_change_carrier (struct net_device * dev, bool new_carrier);
```

Arguments

dev device

new_carrier new value

Description

Change device carrier

Name

dev_get_phys_port_id — Get device physical port ID

Synopsis

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

Arguments

dev device

ppid port ID

Description

Get device physical port ID

Name

dev_get_phys_port_name — Get device physical port name

Synopsis

```
int dev_get_phys_port_name (struct net_device * dev, char * name, size_t
len);
```

Arguments

dev device

name port name

len -- undescribed --

Description

Get device physical port name

Name

netdev_update_features — recalculate device features

Synopsis

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

Arguments

dev the device to check

Description

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

Name

netdev_change_features — recalculate device features

Synopsis

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

Arguments

dev the device to check

Description

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

Name

netif_stacked_transfer_operstate — transfer operstate

Synopsis

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

Arguments

rootdev the root or lower level device to transfer state from

dev the device to transfer operstate to

Description

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

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.

Name

init_dummy_netdev — init a dummy network device for NAPI

Synopsis

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

Arguments

dev device to init

Description

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

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.

Name

dev_get_stats — get network device statistics

Synopsis

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

Arguments

dev device to get statistics from

storage place to store stats

Description

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

Name

alloc_netdev_mqs — allocate network device

Synopsis

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

Arguments

sizeof_priv	size of private data to allocate space for
name	device name format string
name_assign_type	origin of device name
setup	callback to initialize device
txqs	the number of TX subqueues to allocate
rxqs	the number of RX subqueues to allocate

Description

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

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.

Name

synchronize_net — Synchronize with packet receive processing

Synopsis

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

Arguments

void no arguments

Description

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

Name

unregister_netdevice_queue — remove device from the kernel

Synopsis

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

Arguments

dev device

head list

Description

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

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

Name

unregister_netdevice_many — unregister many devices

Synopsis

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

Arguments

head list of devices

Note

As most callers use a stack allocated list_head, we force a `list_del` to make sure stack wont be corrupted later.

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.

Name

dev_change_net_namespace — move device to different nethost namespace

Synopsis

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

Arguments

dev device

net network namespace

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

Description

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

Callers must hold the rtnl semaphore.

Name

netdev_increment_features — increment feature set by one

Synopsis

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

Arguments

all current feature set

one new feature set

mask mask feature set

Description

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

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 int len);
```

Arguments

skb buffer to alter

dev source device

type Ethernet type field

daddr destination address (NULL leave destination address)

saddr source address (NULL use device source address)

len packet length (<= skb->len)

Description

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

Name

eth_get_headlen — determine the the length of header for an ethernet frame

Synopsis

```
u32 eth_get_headlen (void * data, unsigned int len);
```

Arguments

data pointer to start of frame

len total length of frame

Description

Make a best effort attempt to pull the length for all of the headers for a given frame in a linear buffer.

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.

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

Name

eth_header_cache — fill cache entry from neighbour

Synopsis

```
int eth_header_cache (const struct neighbour * neigh, struct hh_cache
* hh, __be16 type);
```

Arguments

neigh source neighbour

hh destination cache entry

type Ethernet type field

Description

Create an Ethernet header template from the neighbour.

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.

Name

eth_prepare_mac_addr_change — prepare for mac change

Synopsis

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

Arguments

dev network device

p socket address

Name

eth_commit_mac_addr_change — commit mac change

Synopsis

```
void eth_commit_mac_addr_change (struct net_device * dev, void * p);
```

Arguments

dev network device

p socket address

Name

eth_mac_addr — set new Ethernet hardware address

Synopsis

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

Arguments

dev network device

p socket address

Description

Change hardware address of device.

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

Name

eth_change_mtu — set new MTU size

Synopsis

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

Arguments

dev network device

new_mtu new Maximum Transfer Unit

Description

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

Name

ether_setup — setup Ethernet network device

Synopsis

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

Arguments

dev network device

Description

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

Name

alloc_etherdev_mqs — Allocates and sets up an Ethernet device

Synopsis

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

Arguments

sizeof_priv	Size of additional driver-private structure to be allocated for this Ethernet device
txqs	The number of TX queues this device has.
rxqs	The number of RX queues this device has.

Description

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

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

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.

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.

Name

is_link_local_ether_addr — Determine if given Ethernet address is link-local

Synopsis

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

Arguments

addr Pointer to a six-byte array containing the Ethernet address

Description

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

Please note

addr must be aligned to u16.

Name

is_zero_ether_addr — Determine if give Ethernet address is all zeros.

Synopsis

```
bool 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.

Please note

addr must be aligned to u16.

Name

is_multicast_ether_addr — Determine if the Ethernet address is a multicast.

Synopsis

```
bool 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.

Name

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

Synopsis

```
bool 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.

Name

is_broadcast_ether_addr — Determine if the Ethernet address is broadcast

Synopsis

```
bool 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.

Please note

addr must be aligned to u16.

Name

is_unicast_ether_addr — Determine if the Ethernet address is unicast

Synopsis

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

Arguments

addr Pointer to a six-byte array containing the Ethernet address

Description

Return true if the address is a unicast address.

Name

is_valid_ether_addr — Determine if the given Ethernet address is valid

Synopsis

```
bool 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.

Please note

addr must be aligned to u16.

Name

eth_random_addr — Generate software assigned random Ethernet address

Synopsis

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

Arguments

addr Pointer to a six-byte array containing the Ethernet address

Description

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

Name

eth_broadcast_addr — Assign broadcast address

Synopsis

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

Arguments

addr Pointer to a six-byte array containing the Ethernet address

Description

Assign the broadcast address to the given address array.

Name

eth_zero_addr — Assign zero address

Synopsis

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

Arguments

addr Pointer to a six-byte array containing the Ethernet address

Description

Assign the zero address to the given address array.

Name

eth_hw_addr_random — Generate software assigned random Ethernet and set device flag

Synopsis

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

Arguments

dev pointer to net_device structure

Description

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

Name

ether_addr_copy — Copy an Ethernet address

Synopsis

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

Arguments

dst Pointer to a six-byte array Ethernet address destination

src Pointer to a six-byte array Ethernet address source

Please note

dst & src must both be aligned to u16.

Name

eth_hw_addr_inherit — Copy dev_addr from another net_device

Synopsis

```
void eth_hw_addr_inherit (struct net_device * dst, struct net_device
* src);
```

Arguments

dst pointer to net_device to copy dev_addr to

src pointer to net_device to copy dev_addr from

Description

Copy the Ethernet address from one net_device to another along with the address attributes (addr_assign_type).

Name

ether_addr_equal — Compare two Ethernet addresses

Synopsis

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

Arguments

addr1 Pointer to a six-byte array containing the Ethernet address

addr2 Pointer other six-byte array containing the Ethernet address

Description

Compare two Ethernet addresses, returns true if equal

Please note

addr1 & addr2 must both be aligned to u16.

Name

ether_addr_equal_64bits — Compare two Ethernet addresses

Synopsis

```
bool ether_addr_equal_64bits (const u8 addr1[6+2], const u8 addr2[6+2]);
```

Arguments

addr1[6+2] Pointer to an array of 8 bytes

addr2[6+2] Pointer to an other array of 8 bytes

Description

Compare two Ethernet addresses, returns true if equal, false otherwise.

The function doesn't need any conditional branches and possibly uses word memory accesses on CPU allowing cheap unaligned memory reads. arrays = { byte1, byte2, byte3, byte4, byte5, byte6, pad1, pad2 }

Please note that alignment of addr1 & addr2 are only guaranteed to be 16 bits.

Name

ether_addr_equal_unaligned — Compare two not u16 aligned Ethernet addresses

Synopsis

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

Arguments

addr1 Pointer to a six-byte array containing the Ethernet address

addr2 Pointer other six-byte array containing the Ethernet address

Description

Compare two Ethernet addresses, returns true if equal

Please note

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

Name

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

Synopsis

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

Arguments

dev	Pointer to a device structure
addr[6 + 2]	Pointer to a six-byte array containing the Ethernet address

Description

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

Note that this function calls `ether_addr_equal_64bits` so take care of the right padding.

Name

compare_ether_header — Compare two Ethernet headers

Synopsis

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

Arguments

a Pointer to Ethernet header

b Pointer to Ethernet header

Description

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

Name

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

Synopsis

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

Arguments

skb Buffer to pad

Description

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

Name

napi_schedule_prep — check if napi can be scheduled

Synopsis

```
bool 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.

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.

Name

napi_schedule_irqoff — schedule NAPI poll

Synopsis

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

Arguments

n napi context

Description

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

Name

napi_complete — NAPI processing complete

Synopsis

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

Arguments

n napi context

Description

Mark NAPI processing as complete. Consider using `napi_complete_done` instead.

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.

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.

Name

enum netdev_priv_flags — struct net_device priv_flags

Synopsis

```
enum netdev_priv_flags {
  IFF_802_1Q_VLAN,
  IFF_EBRIDGE,
  IFF_SLAVE_INACTIVE,
  IFF_MASTER_8023AD,
  IFF_MASTER_ALB,
  IFF_BONDING,
  IFF_SLAVE_NEEDARP,
  IFF_ISATAP,
  IFF_MASTER_ARPMON,
  IFF_WAN_HDLC,
  IFF_XMIT_DST_RELEASE,
  IFF_DONT_BRIDGE,
  IFF_DISABLE_NETPOLL,
  IFF_MACVLAN_PORT,
  IFF_BRIDGE_PORT,
  IFF_OVS_DATAPATH,
  IFF_TX_SKB_SHARING,
  IFF_UNICAST_FLT,
  IFF_TEAM_PORT,
  IFF_SUPP_NOFCS,
  IFF_LIVE_ADDR_CHANGE,
  IFF_MACVLAN,
  IFF_XMIT_DST_RELEASE_PERM,
  IFF_IPVLAN_MASTER,
  IFF_IPVLAN_SLAVE
};
```

Constants

IFF_802_1Q_VLAN	802.1Q VLAN device
IFF_EBRIDGE	Ethernet bridging device
IFF_SLAVE_INACTIVE	bonding slave not the curr. active
IFF_MASTER_8023AD	bonding master, 802.3ad
IFF_MASTER_ALB	bonding master, balance-alb
IFF_BONDING	bonding master or slave
IFF_SLAVE_NEEDARP	need ARPs for validation
IFF_ISATAP	ISATAP interface (RFC4214)
IFF_MASTER_ARPMON	bonding master, ARP mon in use
IFF_WAN_HDLC	WAN HDLC device

IFF_XMIT_DST_RELEASE	`dev_hard_start_xmit` is allowed to release skb->dst
IFF_DONT_BRIDGE	disallow bridging this ether dev
IFF_DISABLE_NETPOLL	disable netpoll at run-time
IFF_MACVLAN_PORT	device used as macvlan port
IFF_BRIDGE_PORT	device used as bridge port
IFF_OVS_DATAPATH	device used as Open vSwitch datapath port
IFF_TX_SKB_SHARING	The interface supports sharing skbs on transmit
IFF_UNICAST_FLT	Supports unicast filtering
IFF_TEAM_PORT	device used as team port
IFF_SUPP_NOFCS	device supports sending custom FCS
IFF_LIVE_ADDR_CHANGE	device supports hardware address change when it's running
IFF_MACVLAN	Macvlan device
IFF_XMIT_DST_RELEASE_PERM	-- undescribed --
IFF_IPVLAN_MASTER	-- undescribed --
IFF_IPVLAN_SLAVE	-- undescribed --

Description

These are the struct net_device, they are only set internally by drivers and used in the kernel. These flags are invisible to userspace, this means that the order of these flags can change during any kernel release.

You should have a pretty good reason to be extending these flags.

Name

struct net_device — The DEVICE structure. Actually, this whole structure is a big mistake. It mixes I/O data with strictly “high-level” data, and it has to know about almost every data structure used in the INET module.

Synopsis

```
struct net_device {
  char name[IFNAMSIZ];
  struct hlist_node name_hlist;
  char * ifalias;
  unsigned long mem_end;
  unsigned long mem_start;
  unsigned long base_addr;
  int irq;
  atomic_t carrier_changes;
  unsigned long state;
  struct list_head dev_list;
  struct list_head napi_list;
  struct list_head unreg_list;
  struct list_head close_list;
  struct {unnamed_struct};
  struct garp_port __rcu * garp_port;
  struct mrp_port __rcu * mrp_port;
  struct device dev;
  const struct attribute_group * sysfs_groups[4];
  const struct attribute_group * sysfs_rx_queue_group;
  const struct rtnl_link_ops * rtnl_link_ops;
#define GSO_MAX_SIZE  65536
  unsigned int gso_max_size;
#define GSO_MAX_SEGS  65535
  u16 gso_max_segs;
  u16 gso_min_segs;
#ifdef CONFIG_DCB
  const struct dcbnl_rtnl_ops * dcbnl_ops;
#endif
  u8 num_tc;
  struct netdev_tc_txq tc_to_txq[TC_MAX_QUEUE];
  u8 prio_tc_map[TC_BITMASK + 1];
#if IS_ENABLED(CONFIG_FCOE)
  unsigned int fcoe_ddp_xid;
#endif
#if IS_ENABLED(CONFIG_CGROUP_NET_PRIO)
  struct netprio_map __rcu * priomap;
#endif
  struct phy_device * phydev;
  struct lock_class_key * qdisc_tx_busylock;
};
```

Members

name[IFNAMSIZ]	This is the first field of the “visible” part of this structure (i.e. as seen by users in the “Space.c” file). It is the name of the interface.
name_hlist	Device name hash chain, please keep it close to name[]
ifalias	SNMP alias
mem_end	Shared memory end
mem_start	Shared memory start
base_addr	Device I/O address
irq	Device IRQ number
carrier_changes	Stats to monitor carrier on<->off transitions
state	Generic network queuing layer state, see netdev_state_t
dev_list	The global list of network devices
napi_list	List entry, that is used for polling napi devices
unreg_list	List entry, that is used, when we are unregistering the device, see the function unregister_netdev
close_list	List entry, that is used, when we are closing the device
{unnamed_struct}	anonymous
garp_port	GARP
mrp_port	MRP
dev	Class/net/name entry
sysfs_groups[4]	Space for optional device, statistics and wireless sysfs groups
sysfs_rx_queue_group	Space for optional per-rx queue attributes
rtnl_link_ops	Rtnl_link_ops
gso_max_size	Maximum size of generic segmentation offload
gso_max_segs	Maximum number of segments that can be passed to the NIC for GSO
gso_min_segs	Minimum number of segments that can be passed to the NIC for GSO
dcbnl_ops	Data Center Bridging netlink ops
num_tc	Number of traffic classes in the net device
tc_to_txq[TC_MAX_QUEUE]	XXX: need comments on this one

prio_tc_map[TC_BITMASK + 1]	need comments on this one
fcoe_ddp_xid	Max exchange id for FCoE LRO by ddp
priomap	XXX: need comments on this one
phydev	Physical device may attach itself for hardware timestamping
qdisc_tx_busylock	XXX: need comments on this one

FIXME

cleanup struct net_device such that network protocol info moves out.

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

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.

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.

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.

Name

netif_queue_stopped — test if transmit queue is flowblocked

Synopsis

```
bool netif_queue_stopped (const struct net_device * dev);
```

Arguments

dev network device

Description

Test if transmit queue on device is currently unable to send.

Name

netdev_txq_bql_enqueue_prefetchw — prefetch bql data for write

Synopsis

```
void netdev_txq_bql_enqueue_prefetchw (struct netdev_queue * dev_queue);
```

Arguments

dev_queue pointer to transmit queue

Description

BQL enabled drivers might use this helper in their `ndo_start_xmit`, to give appropriate hint to the cpu.

Name

netdev_txq_bql_complete_prefetchw — prefetch bql data for write

Synopsis

```
void netdev_txq_bql_complete_prefetchw (struct netdev_queue * dev_queue);
```

Arguments

dev_queue pointer to transmit queue

Description

BQL enabled drivers might use this helper in their TX completion path, to give appropriate hint to the cpu.

Name

netdev_sent_queue — report the number of bytes queued to hardware

Synopsis

```
void netdev_sent_queue (struct net_device * dev, unsigned int bytes);
```

Arguments

dev network device

bytes number of bytes queued to the hardware device queue

Description

Report the number of bytes queued for sending/completion to the network device hardware queue. *bytes* should be a good approximation and should exactly match `netdev_completed_queue` *bytes*

Name

netdev_completed_queue — report bytes and packets completed by device

Synopsis

```
void netdev_completed_queue (struct net_device * dev, unsigned int pkts,
unsigned int bytes);
```

Arguments

dev network device

pkts actual number of packets sent over the medium

bytes actual number of bytes sent over the medium

Description

Report the number of bytes and packets transmitted by the network device hardware queue over the physical medium, *bytes* must exactly match the *bytes* amount passed to netdev_sent_queue

Name

netdev_reset_queue — reset the packets and bytes count of a network device

Synopsis

```
void netdev_reset_queue (struct net_device * dev_queue);
```

Arguments

dev_queue network device

Description

Reset the bytes and packet count of a network device and clear the software flow control OFF bit for this network device

Name

netdev_cap_txqueue — check if selected tx queue exceeds device queues

Synopsis

```
u16 netdev_cap_txqueue (struct net_device * dev, u16 queue_index);
```

Arguments

dev	network device
queue_index	given tx queue index

Description

Returns 0 if given tx queue index >= number of device tx queues, otherwise returns the originally passed tx queue index.

Name

netif_running — test if up

Synopsis

```
bool netif_running (const struct net_device * dev);
```

Arguments

dev network device

Description

Test if the device has been brought up.

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.

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.

Name

__netif_subqueue_stopped — test status of subqueue

Synopsis

```
bool __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.

Name

netif_is_multiqueue — test if device has multiple transmit queues

Synopsis

```
bool netif_is_multiqueue (const struct net_device * dev);
```

Arguments

dev network device

Description

Check if device has multiple transmit queues

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.

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.

Name

netif_carrier_ok — test if carrier present

Synopsis

```
bool netif_carrier_ok (const struct net_device * dev);
```

Arguments

dev network device

Description

Check if carrier is present on device

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.

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.

Name

netif_dormant — test if carrier present

Synopsis

```
bool netif_dormant (const struct net_device * dev);
```

Arguments

dev network device

Description

Check if carrier is present on device

Name

netif_oper_up — test if device is operational

Synopsis

```
bool netif_oper_up (const struct net_device * dev);
```

Arguments

dev network device

Description

Check if carrier is operational

Name

netif_device_present — is device available or removed

Synopsis

```
bool netif_device_present (struct net_device * dev);
```

Arguments

dev network device

Description

Check if device has not been removed from system.

Name

netif_tx_lock — grab network device transmit lock

Synopsis

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

Arguments

dev network device

Description

Get network device transmit lock

Name

__dev_uc_sync — Synchonize device's unicast list

Synopsis

```
int __dev_uc_sync (struct net_device * dev, int (*sync) (struct net_device *, const unsigned char *), int (*unsync) (struct net_device *, const unsigned char *));
```

Arguments

dev device to sync

sync function to call if address should be added

unsync function to call if address should be removed

Description

Add newly added addresses to the interface, and release addresses that have been deleted.

Name

__dev_uc_unsync — Remove synchronized addresses from device

Synopsis

```
void __dev_uc_unsync (struct net_device * dev, int (*unsync) (struct
net_device *, const unsigned char *));
```

Arguments

dev device to sync

unsync function to call if address should be removed

Description

Remove all addresses that were added to the device by dev_uc_sync.

Name

__dev_mc_sync — Synchonize device's multicast list

Synopsis

```
int __dev_mc_sync (struct net_device * dev, int (*sync) (struct net_device *, const unsigned char *), int (*unsync) (struct net_device *, const unsigned char *));
```

Arguments

dev device to sync

sync function to call if address should be added

unsync function to call if address should be removed

Description

Add newly added addresses to the interface, and release addresses that have been deleted.

Name

__dev_mc_unsync — Remove synchronized addresses from device

Synopsis

```
void __dev_mc_unsync (struct net_device * dev, int (*unsync) (struct
net_device *, const unsigned char *));
```

Arguments

dev device to sync

unsync function to call if address should be removed

Description

Remove all addresses that were added to the device by `dev_mc_sync`.

PHY Support

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

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.

Name

phy_mii_ioctl — generic PHY MII ioctl interface

Synopsis

```
int phy_mii_ioctl (struct phy_device * phydev, struct ifreq * ifr, int cmd);
```

Arguments

phydev the phy_device struct

ifr struct ifreq for socket ioctl's

cmd ioctl cmd to execute

Description

Note that this function is currently incompatible with the PHYCONTROL layer. It changes registers without regard to current state. Use at own risk.

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.

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.

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

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

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.

Name

phy_read_mmd_indirect — reads data from the MMD registers

Synopsis

```
int phy_read_mmd_indirect (struct phy_device * phydev, int prtad, int
devad, int addr);
```

Arguments

phydev The PHY device bus

prtad MMD Address

devad MMD DEVAD

addr PHY address on the MII bus

Description

it reads data from the MMD registers (clause 22 to access to clause 45) of the specified phy address.

To read these register we have

1) Write reg 13 // DEVAD 2) Write reg 14 // MMD Address 3) Write reg 13 // MMD Data Command for MMD DEVAD 3) Read reg 14 // Read MMD data

Name

phy_write_mmd_indirect — writes data to the MMD registers

Synopsis

```
void phy_write_mmd_indirect (struct phy_device * phydev, int prtad, int
devad, int addr, u32 data);
```

Arguments

phydev The PHY device

prtad MMD Address

devad MMD DEVAD

addr PHY address on the MII bus

data data to write in the MMD register

Description

Write data from the MMD registers of the specified phy address.

To write these register we have

1) Write reg 13 // DEVAD 2) Write reg 14 // MMD Address 3) Write reg 13 // MMD Data Command for MMD DEVAD 3) Write reg 14 // Write MMD data

Name

phy_init_eee — init and check the EEE feature

Synopsis

```
int phy_init_eee (struct phy_device * phydev, bool clk_stop_enable);
```

Arguments

phydev	target phy_device struct
clk_stop_enable	PHY may stop the clock during LPI

Description

it checks if the Energy-Efficient Ethernet (EEE) is supported by looking at the MMD registers 3.20 and 7.60/61 and it programs the MMD register 3.0 setting the “Clock stop enable” bit if required.

Name

phy_get_eee_err — report the EEE wake error count

Synopsis

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

Arguments

phydev target phy_device struct

Description

it is to report the number of time where the PHY failed to complete its normal wake sequence.

Name

phy_ethtool_get_eee — get EEE supported and status

Synopsis

```
int phy_ethtool_get_eee (struct phy_device * phydev, struct ethtool_eee
* data);
```

Arguments

phydev target phy_device struct

data ethtool_eee data

Description

it reportes the Supported/Advertisement/LP Advertisement capabilities.

Name

phy_ethtool_set_eee — set EEE supported and status

Synopsis

```
int phy_ethtool_set_eee (struct phy_device * phydev, struct ethtool_eee
* data);
```

Arguments

phydev target phy_device struct

data ethtool_eee data

Description

it is to program the Advertisement EEE register.

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.

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 or < 0 on error.

Name

phy_aneg_done — return auto-negotiation status

Synopsis

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

Arguments

phydev target phy_device struct

Description

Return the auto-negotiation status from this *phydev* Returns > 0 on success or < 0 on error. 0 means that auto-negotiation is still pending.

Name

phy_find_setting — find a PHY settings array entry that matches speed & duplex

Synopsis

```
unsigned 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.

Name

phy_find_valid — find a PHY setting that matches the requested features mask

Synopsis

```
unsigned int phy_find_valid (unsigned 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.

Name

phy_check_valid — check if there is a valid PHY setting which matches speed, duplex, and feature mask

Synopsis

```
bool phy_check_valid (int speed, int duplex, u32 features);
```

Arguments

speed	speed to match
duplex	duplex to match
features	A mask of the valid settings

Description

Returns true if there is a valid setting, false otherwise.

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.

Name

phy_start_machine — start PHY state machine tracking

Synopsis

```
void phy_start_machine (struct phy_device * phydev);
```

Arguments

phydev the phy_device struct

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 maintain your own state machine, do not call this function.

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.

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.

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.

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

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

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

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

Name

phy_register_fixup — creates a new phy_fixup and adds it to the list

Synopsis

```
int phy_register_fixup (const char * bus_id, u32 phy_uid, u32 phy_uid_mask, int (*run) (struct phy_device *));
```

Arguments

bus_id	A string which matches phydev->dev.bus_id (or PHY_ANY_ID)
phy_uid	Used to match against phydev->phy_id (the UID of the PHY) It can also be PHY_ANY_UID
phy_uid_mask	Applied to phydev->phy_id and fixup->phy_uid before comparison
run	The actual code to be run when a matching PHY is found

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,
bool is_c45);
```

Arguments

bus the target MII bus

addr PHY address on the MII bus

is_c45 If true the PHY uses the 802.3 clause 45 protocol

Description

Reads the ID registers of the PHY at *addr* on the *bus*, then allocates and returns the phy_device to represent it.

Name

phy_device_register — Register the phy device on the MDIO bus

Synopsis

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

Arguments

phydev phy_device structure to be added to the MDIO bus

Name

phy_find_first — finds the first PHY device on the bus

Synopsis

```
struct phy_device * phy_find_first (struct mii_bus * bus);
```

Arguments

bus the target MII bus

Name

phy_connect_direct — connect an ethernet device to a specific phy_device

Synopsis

```
int phy_connect_direct (struct net_device * dev, struct phy_device
* phydev, void (*handler) (struct net_device *), phy_interface_t
interface);
```

Arguments

dev	the network device to connect
phydev	the pointer to the phy device
handler	callback function for state change notifications
interface	PHY device's interface

Name

phy_connect — connect an ethernet device to a PHY device

Synopsis

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

Arguments

dev	the network device to connect
bus_id	the id string of the PHY device to connect
handler	callback function for state change notifications
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.

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

Name

phy_attach_direct — attach a network device to a given PHY device pointer

Synopsis

```
int phy_attach_direct (struct net_device * dev, struct phy_device *
phydev, u32 flags, phy_interface_t interface);
```

Arguments

dev	network device to attach
phydev	Pointer to phy_device to attach
flags	PHY device's dev_flags
interface	PHY device's interface

Description

Called by drivers to attach to a particular PHY device. The phy_device is found, and properly hooked up to the phy_driver. If no driver is attached, then a generic 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.

Name

phy_attach — attach a network device to a particular PHY device

Synopsis

```
struct phy_device * phy_attach (struct net_device * dev, const char *
bus_id, phy_interface_t interface);
```

Arguments

dev	network device to attach
bus_id	Bus ID of PHY device to attach
interface	PHY device's interface

Description

Same as `phy_attach_direct` except that a PHY bus_id string is passed instead of a pointer to a struct phy_device.

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

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`.

Name

genphy_restart_aneg — Enable and Restart Autonegotiation

Synopsis

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

Arguments

phydev target phy_device struct

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.

Name

genphy_aneg_done — return auto-negotiation status

Synopsis

```
int genphy_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.

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.

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.

Name

genphy_soft_reset — software reset the PHY via BMCR_RESET bit

Synopsis

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

Arguments

phydev target phy_device struct

Description

Perform a software PHY reset using the standard BMCR_RESET bit and poll for the reset bit to be cleared.

Returns

0 on success, < 0 on failure

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

Name

get_phy_c45_ids — reads the specified addr for its 802.3-c45 IDs.

Synopsis

```
int get_phy_c45_ids (struct mii_bus * bus, int addr, u32 * phy_id,
struct phy_c45_device_ids * c45_ids);
```

Arguments

bus	the target MII bus
addr	PHY address on the MII bus
phy_id	where to store the ID retrieved.
c45_ids	where to store the c45 ID information.

Description

If the PHY devices-in-package appears to be valid, it and the corresponding identifiers are stored in *c45_ids*, zero is stored in *phy_id*. Otherwise 0xffffffff is stored in *phy_id*. Returns zero on success.

Name

get_phy_id — reads the specified addr for its ID.

Synopsis

```
int get_phy_id (struct mii_bus * bus, int addr, u32 * phy_id, bool is_c45, struct phy_c45_device_ids * c45_ids);
```

Arguments

bus	the target MII bus
addr	PHY address on the MII bus
phy_id	where to store the ID retrieved.
is_c45	If true the PHY uses the 802.3 clause 45 protocol
c45_ids	where to store the c45 ID information.

Description

In the case of a 802.3-c22 PHY, reads the ID registers of the PHY at *addr* on the *bus*, stores it in *phy_id* and returns zero on success.

In the case of a 802.3-c45 PHY, `get_phy_c45_ids` is invoked, and its return value is in turn returned.

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.

Name

phy_poll_reset — Safely wait until a PHY reset has properly completed

Synopsis

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

Arguments

phydev The PHY device to poll

Description

According to IEEE 802.3, Section 2, Subsection 22.2.4.1.1, as published in 2008, a PHY reset may take up to 0.5 seconds. The MII BMCR register must be polled until the BMCR_RESET bit clears.

Furthermore, any attempts to write to PHY registers may have no effect or even generate MDIO bus errors until this is complete.

Some PHYs (such as the Marvell 88E1111) don't entirely conform to the standard and do not fully reset after the BMCR_RESET bit is set, and may even *REQUIRE* a soft-reset to properly restart autonegotiation. In an effort to support such broken PHYs, this function is separate from the standard `phy_init_hw` which will zero all the other bits in the BMCR and reapply all driver-specific and board-specific fixups.

Name

genphy_config_advert — sanitize and advertise auto-negotiation parameters

Synopsis

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

Arguments

phydev target phy_device struct

Description

Writes MII_ADVERTISE with the appropriate values, after sanitizing the values to make sure we only advertise what is supported. Returns < 0 on error, 0 if the PHY's advertisement hasn't changed, and > 0 if it has changed.

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).

Name

mdiobus_alloc_size — allocate a mii_bus structure

Synopsis

```
struct mii_bus * mdiobus_alloc_size (size_t size);
```

Arguments

size extra amount of memory to allocate for private storage. If non-zero, then bus->priv is points to that memory.

Description

called by a bus driver to allocate an mii_bus structure to fill in.

Name

devm_mdiobus_alloc_size — Resource-managed `mdiobus_alloc_size`

Synopsis

```
struct mii_bus * devm_mdiobus_alloc_size (struct device * dev, int sizeof_priv);
```

Arguments

dev	Device to allocate mii_bus for
sizeof_priv	Space to allocate for private structure.

Description

Managed mdiobus_alloc_size. mii_bus allocated with this function is automatically freed on driver detach.

If an mii_bus allocated with this function needs to be freed separately, `devm_mdiobus_free` must be used.

RETURNS

Pointer to allocated mii_bus on success, NULL on failure.

Name

devm_mdiobus_free — Resource-managed `mdiobus_free`

Synopsis

```
void devm_mdiobus_free (struct device * dev, struct mii_bus * bus);
```

Arguments

`dev` Device this mii_bus belongs to

`bus` the mii_bus associated with the device

Description

Free mii_bus allocated with `devm_mdiobus_alloc_size`.

Name

of_mdio_find_bus — Given an mii_bus node, find the mii_bus.

Synopsis

```
struct mii_bus * of_mdio_find_bus (struct device_node * mdio_bus_np);
```

Arguments

mdio_bus_np Pointer to the mii_bus.

Description

Returns a pointer to the mii_bus, or NULL if none found.

Because the association of a device_node and mii_bus is made via `of_mdiobus_register`, the mii_bus cannot be found before it is registered with `of_mdiobus_register`.

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.

Name

mdiobus_free — free a struct mii_bus

Synopsis

```
void mdiobus_free (struct mii_bus * bus);
```

Arguments

bus mii_bus to free

Description

This function releases the reference to the underlying device object in the mii_bus. If this is the last reference, the mii_bus will be freed.

Name

mdiobus_read — Convenience function for reading a given MII mgmt register

Synopsis

```
int mdiobus_read (struct mii_bus * bus, int addr, u32 regnum);
```

Arguments

bus the mii_bus struct

addr the phy address

regnum register number to read

NOTE

MUST NOT be called from interrupt context, because the bus read/write functions may wait for an interrupt to conclude the operation.

Name

mdiobus_write — Convenience function for writing a given MII mgmt register

Synopsis

```
int mdiobus_write (struct mii_bus * bus, int addr, u32 regnum, u16 val);
```

Arguments

bus	the mii_bus struct
addr	the phy address
regnum	register number to write
val	value to write to *regnum*

NOTE

MUST NOT be called from interrupt context, because the bus read/write functions may wait for an interrupt to conclude the operation.

Name

mdiobus_release — mii_bus device release callback

Synopsis

```
void mdiobus_release (struct device * d);
```

Arguments

d the target struct device that contains the mii_bus

Description

called when the last reference to an mii_bus is dropped, to free the underlying memory.

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.