Linux Cross Reference

Free Electrons

Embedded Linux Experts

• *source navigation* • diff markup • identifier search • freetext search •

Version:
2.0.40 2.2.26 2.4.37 3.6 3.7 3.8 3.9 3.10 3.11 3.12 3.13 3.14 3.15 3.16 3.17 3.18 3.19 4.0 4.1 ***4.2***

Linux/include/net/inet_sock.h

```
/*
 * INET         An implementation of the TCP/IP protocol suite for the LINUX
 *              operating system.  INET is implemented using the  BSD Socket
 *              interface as the means of communication with the user level.
 *
 *              Definitions for inet_sock
 *
 * Authors:     Many, reorganised here by
 *              Arnaldo Carvalho de Melo <acme@mandriva.com>
 *
 *              This program 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.
 */
#ifndef _INET_SOCK_H
#define _INET_SOCK_H

#include <linux/bitops.h>
#include <linux/kmemcheck.h>
#include <linux/string.h>
#include <linux/types.h>
#include <linux/jhash.h>
#include <linux/netdevice.h>

#include <net/flow.h>
#include <net/sock.h>
#include <net/request_sock.h>
#include <net/netns/hash.h>
#include <net/tcp_states.h>

/** struct ip_options - IP Options
 *
 * @faddr - Saved first hop address
 * @nexthop - Saved nexthop address in LSRR and SSRR
 * @is_strictroute - Strict source route
 * @srr_is_hit - Packet destination addr was our one
 * @is_changed - IP checksum more not valid
 * @rr_needaddr - Need to record addr of outgoing dev
 * @ts_needtime - Need to record timestamp
 * @ts_needaddr - Need to record addr of outgoing dev
 */
struct ip_options {
```

```c
        __be32          faddr;
        __be32          nexthop;
        unsigned char   optlen;
        unsigned char   srr;
        unsigned char   rr;
        unsigned char   ts;
        unsigned char   is_strictroute:1,
                        srr_is_hit:1,
                        is_changed:1,
                        rr_needaddr:1,
                        ts_needtime:1,
                        ts_needaddr:1;
        unsigned char   router_alert;
        unsigned char   cipso;
        unsigned char   __pad2;
        unsigned char   __data[0];
};

struct ip_options_rcu {
        struct rcu_head rcu;
        struct ip_options opt;
};

struct ip_options_data {
        struct ip_options_rcu   opt;
        char                    data[40];
};

struct inet_request_sock {
        struct request_sock     req;
#define ir_loc_addr             req.__req_common.skc_rcv_saddr
#define ir_rmt_addr             req.__req_common.skc_daddr
#define ir_num                  req.__req_common.skc_num
#define ir_rmt_port             req.__req_common.skc_dport
#define ir_v6_rmt_addr          req.__req_common.skc_v6_daddr
#define ir_v6_loc_addr          req.__req_common.skc_v6_rcv_saddr
#define ir_iif                  req.__req_common.skc_bound_dev_if
#define ir_cookie               req.__req_common.skc_cookie
#define ireq_net                req.__req_common.skc_net
#define ireq_state              req.__req_common.skc_state
#define ireq_family             req.__req_common.skc_family

        kmemcheck_bitfield_begin(flags);
        u16                     snd_wscale : 4,
                                rcv_wscale : 4,
                                tstamp_ok  : 1,
                                sack_ok    : 1,
                                wscale_ok  : 1,
                                ecn_ok     : 1,
                                acked      : 1,
                                no_srccheck: 1;
        kmemcheck_bitfield_end(flags);
        u32                     ir_mark;
        union {
                struct ip_options_rcu   *opt;
                struct sk_buff          *pktopts;
        };
};

static inline struct inet_request_sock *inet_rsk(const struct request_sock *sk)
{
        return (struct inet_request_sock *)sk;
}

static inline u32 inet_request_mark(const struct sock *sk, struct sk_buff *skb)
```

```
{
        if (!sk->sk_mark && sock_net(sk)->ipv4.sysctl_tcp_fwmark_accept)
                return skb->mark;

        return sk->sk_mark;
}

struct inet_cork {
        unsigned int            flags;
        __be32                  addr;
        struct ip_options       *opt;
        unsigned int            fragsize;
        int                     length; /* Total length of all frames */
        struct dst_entry        *dst;
        u8                      tx_flags;
        __u8                    ttl;
        __s16                   tos;
        char                    priority;
};

struct inet_cork_full {
        struct inet_cork        base;
        struct flowi            fl;
};

struct ip_mc_socklist;
struct ipv6_pinfo;
struct rtable;

/** struct inet_sock - representation of INET sockets
 *
 * @sk - ancestor class
 * @pinet6 - pointer to IPv6 control block
 * @inet_daddr - Foreign IPv4 addr
 * @inet_rcv_saddr - Bound local IPv4 addr
 * @inet_dport - Destination port
 * @inet_num - Local port
 * @inet_saddr - Sending source
 * @uc_ttl - Unicast TTL
 * @inet_sport - Source port
 * @inet_id - ID counter for DF pkts
 * @tos - TOS
 * @mc_ttl - Multicasting TTL
 * @is_icsk - is this an inet_connection_sock?
 * @uc_index - Unicast outgoing device index
 * @mc_index - Multicast device index
 * @mc_list - Group array
 * @cork - info to build ip hdr on each ip frag while socket is corked
 */
struct inet_sock {
        /* sk and pinet6 has to be the first two members of inet_sock */
        struct sock             sk;
#if IS_ENABLED(CONFIG_IPV6)
        struct ipv6_pinfo       *pinet6;
#endif
        /* Socket demultiplex comparisons on incoming packets. */
#define inet_daddr              sk.__sk_common.skc_daddr
#define inet_rcv_saddr          sk.__sk_common.skc_rcv_saddr
#define inet_dport              sk.__sk_common.skc_dport
#define inet_num                sk.__sk_common.skc_num

        __be32                  inet_saddr;
        __s16                   uc_ttl;
        __u16                   cmsg_flags;
        __be16                  inet_sport;
        __u16                   inet_id;

        struct ip_options_rcu __rcu     *inet_opt;
        int                     rx_dst_ifindex;
        __u8                    tos;
        __u8                    min_ttl;
        __u8                    mc_ttl;
        __u8                    pmtudisc;
        __u8                    recverr:1,
                                is_icsk:1,
                                freebind:1,
                                hdrincl:1,
                                mc_loop:1,
                                transparent:1,
                                mc_all:1,
                                nodefrag:1;
        __u8                    bind_address_no_port:1;
        __u8                    rcv_tos;
        __u8                    convert_csum;
        int                     uc_index;
        int                     mc_index;
        __be32                  mc_addr;
        struct ip_mc_socklist __rcu     *mc_list;
        struct inet_cork_full   cork;
};

#define IPCORK_OPT      1       /* ip-options has been held in ipcork.opt */
#define IPCORK_ALLFRAG  2       /* always fragment (for ipv6 for now) */

/* cmsg flags for inet */
#define IP_CMSG_PKTINFO         BIT(0)
#define IP_CMSG_TTL             BIT(1)
#define IP_CMSG_TOS             BIT(2)
#define IP_CMSG_RECVOPTS        BIT(3)
#define IP_CMSG_RETOPTS         BIT(4)
#define IP_CMSG_PASSSEC         BIT(5)
#define IP_CMSG_ORIGDSTADDR     BIT(6)
#define IP_CMSG_CHECKSUM        BIT(7)

static inline struct inet_sock *inet_sk(const struct sock *sk)
{
        return (struct inet_sock *)sk;
}

static inline void __inet_sk_copy_descendant(struct sock *sk_to,
                                             const struct sock *sk_from,
                                             const int ancestor_size)
{
        memcpy(inet_sk(sk_to) + 1, inet_sk(sk_from) + 1,
               sk_from->sk_prot->obj_size - ancestor_size);
}
#if !(IS_ENABLED(CONFIG_IPV6))
static inline void inet_sk_copy_descendant(struct sock *sk_to,
                                           const struct sock *sk_from)
{
        __inet_sk_copy_descendant(sk_to, sk_from, sizeof(struct inet_sock));
}
#endif

int inet_sk_rebuild_header(struct sock *sk);

static inline unsigned int __inet_ehashfn(const __be32 laddr,
                                          const __u16 lport,
                                          const __be32 faddr,
                                          const __be16 fport,
                                          u32 initval)
{
        return jhash_3words((__force __u32) laddr,
                            (__force __u32) faddr,
                            ((__u32) lport) << 16 | (__force __u32)fport,
                            initval);
}

struct request_sock *inet_reqsk_alloc(const struct request_sock_ops *ops,
                                      struct sock *sk_listener);

static inline __u8 inet_sk_flowi_flags(const struct sock *sk)
{
        __u8 flags = 0;

        if (inet_sk(sk)->transparent || inet_sk(sk)->hdrincl)
                flags |= FLOWI_FLAG_ANYSRC;
        return flags;
}

static inline void inet_inc_convert_csum(struct sock *sk)
{
        inet_sk(sk)->convert_csum++;
}

static inline void inet_dec_convert_csum(struct sock *sk)
{
        if (inet_sk(sk)->convert_csum > 0)
                inet_sk(sk)->convert_csum--;
}

static inline bool inet_get_convert_csum(struct sock *sk)
{
        return !!inet_sk(sk)->convert_csum;
}

#endif  /* _INET_SOCK_H */

```

- Home
- Development
- Services
- Training
- Docs
- Community
- Company
- Blog