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/uapi/linux/if_packet.h

```
 1 #ifndef __LINUX_IF_PACKET_H
 2 #define __LINUX_IF_PACKET_H
 3
 4 #include <linux/types.h>
 5
 6 struct sockaddr_pkt {
 7         unsigned short spkt_family;
 8         unsigned char spkt_device[14];
 9         __be16 spkt_protocol;
10 };
11
12 struct sockaddr_ll {
13         unsigned short  sll_family;
14         __be16          sll_protocol;
15         int             sll_ifindex;
16         unsigned short  sll_hatype;
17         unsigned char   sll_pkttype;
18         unsigned char   sll_halen;
19         unsigned char   sll_addr[8];
20 };
21
22 /* Packet types */
23
24 #define PACKET_HOST             0               /* To us                */
25 #define PACKET_BROADCAST        1               /* To all               */
26 #define PACKET_MULTICAST        2               /* To group             */
27 #define PACKET_OTHERHOST        3               /* To someone else      */
28 #define PACKET_OUTGOING         4               /* Outgoing of any type */
29 #define PACKET_LOOPBACK         5               /* MC/BRD frame looped back */
30 #define PACKET_USER             6               /* To user space        */
31 #define PACKET_KERNEL           7               /* To kernel space      */
32 /* Unused, PACKET_FASTROUTE and PACKET_LOOPBACK are invisible to user space */
33 #define PACKET_FASTROUTE        6               /* Fastrouted frame     */
34
35 /* Packet socket options */
36
37 #define PACKET_ADD_MEMBERSHIP           1
38 #define PACKET_DROP_MEMBERSHIP          2
39 #define PACKET_RECV_OUTPUT              3
40 /* Value 4 is still used by obsolete turbo-packet. */
41 #define PACKET_RX_RING                  5
42 #define PACKET_STATISTICS               6
43 #define PACKET_COPY_THRESH              7
44 #define PACKET_AUXDATA                  8
45 #define PACKET_ORIGDEV                  9
46 #define PACKET_VERSION                  10
47 #define PACKET_HDRLEN                   11
48 #define PACKET_RESERVE                  12
49 #define PACKET_TX_RING                  13
50 #define PACKET_LOSS                     14
51 #define PACKET_VNET_HDR                 15
52 #define PACKET_TX_TIMESTAMP             16
53 #define PACKET_TIMESTAMP                17
54 #define PACKET_FANOUT                   18
55 #define PACKET_TX_HAS_OFF               19
56 #define PACKET_QDISC_BYPASS             20
```

```
57 #define PACKET_ROLLOVER_STATS           21
58
59 #define PACKET_FANOUT_HASH              0
60 #define PACKET_FANOUT_LB                1
61 #define PACKET_FANOUT_CPU               2
62 #define PACKET_FANOUT_ROLLOVER          3
63 #define PACKET_FANOUT_RND               4
64 #define PACKET_FANOUT_QM                5
65 #define PACKET_FANOUT_FLAG_ROLLOVER     0x1000
66 #define PACKET_FANOUT_FLAG_DEFRAG       0x8000
67
68 struct tpacket_stats {
69         unsigned int    tp_packets;
70         unsigned int    tp_drops;
71 };
72
73 struct tpacket_stats_v3 {
74         unsigned int    tp_packets;
75         unsigned int    tp_drops;
76         unsigned int    tp_freeze_q_cnt;
77 };
78
79 struct tpacket_rollover_stats {
80         __aligned_u64   tp_all;
81         __aligned_u64   tp_huge;
82         __aligned_u64   tp_failed;
83 };
84
85 union tpacket_stats_u {
86         struct tpacket_stats stats1;
87         struct tpacket_stats_v3 stats3;
88 };
89
90 struct tpacket_auxdata {
91         __u32           tp_status;
92         __u32           tp_len;
93         __u32           tp_snaplen;
94         __u16           tp_mac;
95         __u16           tp_net;
96         __u16           tp_vlan_tci;
97         __u16           tp_vlan_tpid;
98 };
99
100 /* Rx ring - header status */
101 #define TP_STATUS_KERNEL                    0
102 #define TP_STATUS_USER                  (1 << 0)
103 #define TP_STATUS_COPY                  (1 << 1)
104 #define TP_STATUS_LOSING                (1 << 2)
105 #define TP_STATUS_CSUMNOTREADY          (1 << 3)
106 #define TP_STATUS_VLAN_VALID            (1 << 4) /* auxdata has valid tp_vlan_tci */
107 #define TP_STATUS_BLK_TMO               (1 << 5)
108 #define TP_STATUS_VLAN_TPID_VALID       (1 << 6) /* auxdata has valid tp_vlan_tpid */
109 #define TP_STATUS_CSUM_VALID            (1 << 7)
110
111 /* Tx ring - header status */
112 #define TP_STATUS_AVAILABLE           0
113 #define TP_STATUS_SEND_REQUEST  (1 << 0)
114 #define TP_STATUS_SENDING       (1 << 1)
115 #define TP_STATUS_WRONG_FORMAT  (1 << 2)
116
117 /* Rx and Tx ring - header status */
118 #define TP_STATUS_TS_SOFTWARE           (1 << 29)
119 #define TP_STATUS_TS_SYS_HARDWARE       (1 << 30) /* deprecated, never set */
120 #define TP_STATUS_TS_RAW_HARDWARE       (1 << 31)
121
122 /* Rx ring - feature request bits */
123 #define TP_FT_REQ_FILL_RXHASH   0x1
124
125 struct tpacket_hdr {
126         unsigned long   tp_status;
127         unsigned int    tp_len;
128         unsigned int    tp_snaplen;
129         unsigned short  tp_mac;
130         unsigned short  tp_net;
131         unsigned int    tp_sec;
132         unsigned int    tp_usec;
```

```
133 };
134
135 #define TPACKET_ALIGNMENT       16
136 #define TPACKET_ALIGN(x)        (((x)+TPACKET_ALIGNMENT-1)&~(TPACKET_ALIGNMENT-1))
137 #define TPACKET_HDRLEN          (TPACKET_ALIGN(sizeof(struct tpacket_hdr)) + sizeof(struct sockaddr_ll))
138
139 struct tpacket2_hdr {
140         __u32           tp_status;
141         __u32           tp_len;
142         __u32           tp_snaplen;
143         __u16           tp_mac;
144         __u16           tp_net;
145         __u32           tp_sec;
146         __u32           tp_nsec;
147         __u16           tp_vlan_tci;
148         __u16           tp_vlan_tpid;
149         __u8            tp_padding[4];
150 };
151
152 struct tpacket_hdr_variant1 {
153         __u32   tp_rxhash;
154         __u32   tp_vlan_tci;
155         __u16   tp_vlan_tpid;
156         __u16   tp_padding;
157 };
158
159 struct tpacket3_hdr {
160         __u32           tp_next_offset;
161         __u32           tp_sec;
162         __u32           tp_nsec;
163         __u32           tp_snaplen;
164         __u32           tp_len;
165         __u32           tp_status;
166         __u16           tp_mac;
167         __u16           tp_net;
168         /* pkt_hdr variants */
169         union {
170                 struct tpacket_hdr_variant1 hv1;
171         };
172         __u8            tp_padding[8];
173 };
174
175 struct tpacket_bd_ts {
176         unsigned int ts_sec;
177         union {
178                 unsigned int ts_usec;
179                 unsigned int ts_nsec;
180         };
181 };
182
183 struct tpacket_hdr_v1 {
184         __u32   block_status;
185         __u32   num_pkts;
186         __u32   offset_to_first_pkt;
187
188         /* Number of valid bytes (including padding)
189          * blk_len <= tp_block_size
190          */
191         __u32   blk_len;
192
193         /*
194          * Quite a few uses of sequence number:
195          * 1. Make sure cache flush etc worked.
196          *    Well, one can argue - why not use the increasing ts below?
197          *    But look at 2. below first.
198          * 2. When you pass around blocks to other user space decoders,
199          *    you can see which blk[s] is[are] outstanding etc.
200          * 3. Validate kernel code.
201          */
202         __aligned_u64   seq_num;
203
204         /*
205          * ts_last_pkt:
206          *
207          * Case 1.      Block has 'N'(N >=1) packets and TMO'd(timed out)
208          *              ts_last_pkt == 'time-stamp of last packet' and NOT the
```

```
209          *              time when the timer fired and the block was closed.
210          *              By providing the ts of the last packet we can absolutely
211          *              guarantee that time-stamp wise, the first packet in the
212          *              next block will never precede the last packet of the
213          *              previous block.
214          * Case 2.      Block has zero packets and TMO'd
215          *              ts_last_pkt = time when the timer fired and the block
216          *              was closed.
217          * Case 3.      Block has 'N' packets and NO TMO.
218          *              ts_last_pkt = time-stamp of the last pkt in the block.
219          *
220          * ts_first_pkt:
221          *              Is always the time-stamp when the block was opened.
222          *              Case a) ZERO packets
223          *                      No packets to deal with but atleast you know the
224          *                      time-interval of this block.
225          *              Case b) Non-zero packets
226          *                      Use the ts of the first packet in the block.
227          *
228          */
229         struct tpacket_bd_ts    ts_first_pkt, ts_last_pkt;
230 };
231
232 union tpacket_bd_header_u {
233         struct tpacket_hdr_v1 bh1;
234 };
235
236 struct tpacket_block_desc {
237         __u32 version;
238         __u32 offset_to_priv;
239         union tpacket_bd_header_u hdr;
240 };
241
242 #define TPACKET2_HDRLEN         (TPACKET_ALIGN(sizeof(struct tpacket2_hdr)) + sizeof(struct sockaddr_ll))
243 #define TPACKET3_HDRLEN         (TPACKET_ALIGN(sizeof(struct tpacket3_hdr)) + sizeof(struct sockaddr_ll))
244
245 enum tpacket_versions {
246         TPACKET_V1,
247         TPACKET_V2,
248         TPACKET_V3
249 };
250
251 /*
252    Frame structure:
253
254    - Start. Frame must be aligned to TPACKET_ALIGNMENT=16
255    - struct tpacket_hdr
256    - pad to TPACKET_ALIGNMENT=16
257    - struct sockaddr_ll
258    - Gap, chosen so that packet data (Start+tp_net) alignes to TPACKET_ALIGNMENT=16
259    - Start+tp_mac: [ Optional MAC header ]
260    - Start+tp_net: Packet data, aligned to TPACKET_ALIGNMENT=16.
261    - Pad to align to TPACKET_ALIGNMENT=16
262  */
263
264 struct tpacket_req {
265         unsigned int    tp_block_size;  /* Minimal size of contiguous block */
266         unsigned int    tp_block_nr;    /* Number of blocks */
267         unsigned int    tp_frame_size;  /* Size of frame */
268         unsigned int    tp_frame_nr;    /* Total number of frames */
269 };
270
271 struct tpacket_req3 {
272         unsigned int    tp_block_size;  /* Minimal size of contiguous block */
273         unsigned int    tp_block_nr;    /* Number of blocks */
274         unsigned int    tp_frame_size;  /* Size of frame */
275         unsigned int    tp_frame_nr;    /* Total number of frames */
276         unsigned int    tp_retire_blk_tov; /* timeout in msecs */
277         unsigned int    tp_sizeof_priv; /* offset to private data area */
278         unsigned int    tp_feature_req_word;
279 };
280
281 union tpacket_req_u {
282         struct tpacket_req      req;
283         struct tpacket_req3     req3;
284 };
```

```

struct packet_mreq {
        int             mr_ifindex;
        unsigned short  mr_type;
        unsigned short  mr_alen;
        unsigned char   mr_address[8];
};

#define PACKET_MR_MULTICAST     0
#define PACKET_MR_PROMISC       1
#define PACKET_MR_ALLMULTI      2
#define PACKET_MR_UNICAST       3

#endif

```