

```

*****
30228 Wed Aug  8 12:51:10 2012
new/usr/src/uts/common/inet/tcp/tcp_opt_data.c
3065 some functions in the tcp module can be static
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
23 * Copyright (c) 2011 Nexenta Systems, Inc. All rights reserved.
24 */

26 #include <sys/types.h>
27 #include <sys/stream.h>
28 #define _SUN_TPI_VERSION 2
29 #include <sys/tihdr.h>
30 #include <sys/socket.h>
31 #include <sys/xti_xtiopt.h>
32 #include <sys/xti_inet.h>
33 #include <sys/policy.h>

35 #include <inet/common.h>
36 #include <netinet/ip6.h>
37 #include <netinet/ip.h>

39 #include <netinet/in.h>
40 #include <netinet/tcp.h>
41 #include <inet/optcom.h>
42 #include <inet/proto_set.h>
43 #include <inet/tcp_impl.h>

45 static int    tcp_opt_default(queue_t *, int, int, uchar_t *);

47 #endif /* ! codereview */
48 /*
49  * Table of all known options handled on a TCP protocol stack.
50  *
51  * Note: This table contains options processed by both TCP and IP levels
52  * and is the superset of options that can be performed on a TCP over IP
53  * stack.
54  */
55 opdes_t tcp_opt_arr[] = {

57 { SO_LINGER,    SOL_SOCKET, OA_RW, OA_RW, OP_NP, 0,
58   sizeof (struct linger), 0 },

60 { SO_DEBUG,    SOL_SOCKET, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },
61 { SO_KEEPAIVE, SOL_SOCKET, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },

```

```

62 { SO_DONTROUTE, SOL_SOCKET, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },
63 { SO_USELOOPBACK, SOL_SOCKET, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },
64 },
65 { SO_BROADCAST, SOL_SOCKET, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },
66 { SO_REUSEADDR, SOL_SOCKET, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },
67 { SO_OOBINLINE, SOL_SOCKET, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },
68 { SO_TYPE,      SOL_SOCKET, OA_R,  OA_R,  OP_NP, 0, sizeof (int), 0 },
69 { SO_SNDBUF,    SOL_SOCKET, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },
70 { SO_RCVBUF,    SOL_SOCKET, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },
71 { SO_SNDTIMEO,  SOL_SOCKET, OA_RW, OA_RW, OP_NP, 0,
72   sizeof (struct timeval), 0 },
73 { SO_RCVTIMEO,  SOL_SOCKET, OA_RW, OA_RW, OP_NP, 0,
74   sizeof (struct timeval), 0 },
75 { SO_DGRAM_ERRIND, SOL_SOCKET, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },
76 },
77 { SO_SND_COPYAVOID, SOL_SOCKET, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },
78 { SO_ANON_MLP, SOL_SOCKET, OA_RW, OA_RW, OP_NP, 0, sizeof (int),
79   0 },
80 { SO_MAC_EXEMPT, SOL_SOCKET, OA_RW, OA_RW, OP_NP, 0, sizeof (int),
81   0 },
82 { SO_MAC_IMPLICIT, SOL_SOCKET, OA_RW, OA_RW, OP_NP, 0, sizeof (int),
83   0 },
84 { SO_ALLZONES, SOL_SOCKET, OA_R,  OA_RW, OP_CONFIG, 0, sizeof (int),
85   0 },
86 { SO_EXCLBIND, SOL_SOCKET, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },

88 { SO_DOMAIN,    SOL_SOCKET, OA_R,  OA_R,  OP_NP, 0, sizeof (int), 0 },

90 { SO_PROTOTYPE, SOL_SOCKET, OA_R,  OA_R,  OP_NP, 0, sizeof (int), 0 },

92 { TCP_NODELAY,  IPPROTO_TCP, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },
93 },
94 { TCP_MAXSEG,   IPPROTO_TCP, OA_R,  OA_R,  OP_NP, 0, sizeof (uint_t),
95   536 },

97 { TCP_NOTIFY_THRESHOLD, IPPROTO_TCP, OA_RW, OA_RW, OP_NP,
98   OP_DEF_FN, sizeof (int), -1 /* not initialized */ },

100 { TCP_ABORT_THRESHOLD, IPPROTO_TCP, OA_RW, OA_RW, OP_NP,
101   OP_DEF_FN, sizeof (int), -1 /* not initialized */ },

103 { TCP_CONN_NOTIFY_THRESHOLD, IPPROTO_TCP, OA_RW, OA_RW, OP_NP,
104   OP_DEF_FN, sizeof (int), -1 /* not initialized */ },

106 { TCP_CONN_ABORT_THRESHOLD, IPPROTO_TCP, OA_RW, OA_RW, OP_NP,
107   OP_DEF_FN, sizeof (int), -1 /* not initialized */ },

109 { TCP_RECVDSTADDR, IPPROTO_TCP, OA_RW, OA_RW, OP_NP, 0, sizeof (int),
110   0 },

112 { TCP_ANONPRIVBIND, IPPROTO_TCP, OA_R,  OA_RW, OP_PRIVPORT, 0,
113   sizeof (int), 0 },

115 { TCP_EXCLBIND, IPPROTO_TCP, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },
116 },

118 { TCP_INIT_CWND, IPPROTO_TCP, OA_RW, OA_RW, OP_CONFIG, 0,
119   sizeof (int), 0 },

121 { TCP_KEEPAIVE_THRESHOLD, IPPROTO_TCP, OA_RW, OA_RW, OP_NP, 0,
122   sizeof (int), 0 },

124 { TCP_KEEPIDLE, IPPROTO_TCP, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },

126 { TCP_KEEPCNT, IPPROTO_TCP, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },

```

```

128 { TCP_KEEPIPTVL, IPPROTO_TCP, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },
130 { TCP_KEEPA_LIVE_ABORT_THRESHOLD, IPPROTO_TCP, OA_RW, OA_RW, OP_NP, 0,
131     sizeof (int), 0 },
133 { TCP_CORK, IPPROTO_TCP, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },
135 { TCP_RTO_INITIAL, IPPROTO_TCP, OA_RW, OA_RW, OP_NP, 0, sizeof (uint32_t), 0 },
137 { TCP_RTO_MIN, IPPROTO_TCP, OA_RW, OA_RW, OP_NP, 0, sizeof (uint32_t), 0 },
139 { TCP_RTO_MAX, IPPROTO_TCP, OA_RW, OA_RW, OP_NP, 0, sizeof (uint32_t), 0 },
141 { TCP_LINGER2, IPPROTO_TCP, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },
143 { IP_OPTIONS,    IPPROTO_IP, OA_RW, OA_RW, OP_NP,
144     (OP_VARLEN|OP_NODEFAULT),
145     IP_MAX_OPT_LENGTH + IP_ADDR_LEN, -1 /* not initialized */ },
146 { T_IP_OPTIONS,  IPPROTO_IP, OA_RW, OA_RW, OP_NP,
147     (OP_VARLEN|OP_NODEFAULT),
148     IP_MAX_OPT_LENGTH + IP_ADDR_LEN, -1 /* not initialized */ },
150 { IP_TOS,        IPPROTO_IP, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },
151 { T_IP_TOS,      IPPROTO_IP, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },
152 { IP_TTL,        IPPROTO_IP, OA_RW, OA_RW, OP_NP, OP_DEF_FN,
153     sizeof (int), -1 /* not initialized */ },
155 { IP_SEC_OPT,   IPPROTO_IP, OA_RW, OA_RW, OP_NP, OP_NODEFAULT,
156     sizeof (ipsec_req_t), -1 /* not initialized */ },
158 { IP_BOUND_IF, IPPROTO_IP, OA_RW, OA_RW, OP_NP, 0,
159     sizeof (int), 0 /* no ifindex */ },
161 { IP_UNSPEC_SRC, IPPROTO_IP, OA_R, OA_RW, OP_RAW, 0,
162     sizeof (int), 0 },
164 { IPV6_UNICAST_HOPS, IPPROTO_IPV6, OA_RW, OA_RW, OP_NP, OP_DEF_FN,
165     sizeof (int), -1 /* not initialized */ },
167 { IPV6_BOUND_IF, IPPROTO_IPV6, OA_RW, OA_RW, OP_NP, 0,
168     sizeof (int), 0 /* no ifindex */ },
170 { IP_DONTFRAG,  IPPROTO_IP, OA_RW, OA_RW, OP_NP, 0, sizeof (int), 0 },
172 { IP_NEXTHOP,   IPPROTO_IP, OA_R, OA_RW, OP_CONFIG, 0,
173     sizeof (in_addr_t), -1 /* not initialized */ },
175 { IPV6_UNSPEC_SRC, IPPROTO_IPV6, OA_R, OA_RW, OP_RAW, 0,
176     sizeof (int), 0 },
178 { IPV6_PKTINFO, IPPROTO_IPV6, OA_RW, OA_RW, OP_NP,
179     (OP_NODEFAULT|OP_VARLEN),
180     sizeof (struct in6_pktinfo), -1 /* not initialized */ },
181 { IPV6_NEXTHOP,  IPPROTO_IPV6, OA_RW, OA_RW, OP_NP,
182     OP_NODEFAULT,
183     sizeof (sin6_t), -1 /* not initialized */ },
184 { IPV6_HOPOPTS,  IPPROTO_IPV6, OA_RW, OA_RW, OP_NP,
185     (OP_VARLEN|OP_NODEFAULT), 255*8,
186     -1 /* not initialized */ },
187 { IPV6_DSTOPTS,  IPPROTO_IPV6, OA_RW, OA_RW, OP_NP,
188     (OP_VARLEN|OP_NODEFAULT), 255*8,
189     -1 /* not initialized */ },
190 { IPV6_RTHDRDSTOPTS, IPPROTO_IPV6, OA_RW, OA_RW, OP_NP,
191     (OP_VARLEN|OP_NODEFAULT), 255*8,
192     -1 /* not initialized */ },
193 { IPV6_RTHDR,    IPPROTO_IPV6, OA_RW, OA_RW, OP_NP,

```

```

194     (OP_VARLEN|OP_NODEFAULT), 255*8,
195     -1 /* not initialized */ },
196 { IPV6_TCLASS,  IPPROTO_IPV6, OA_RW, OA_RW, OP_NP,
197     OP_NODEFAULT,
198     sizeof (int), -1 /* not initialized */ },
199 { IPV6_PATHMTU, IPPROTO_IPV6, OA_RW, OA_RW, OP_NP,
200     OP_NODEFAULT,
201     sizeof (struct ip6_mtuinfo), -1 /* not initialized */ },
202 { IPV6_DONTFRAG, IPPROTO_IPV6, OA_RW, OA_RW, OP_NP, 0,
203     sizeof (int), 0 },
204 { IPV6_USE_MIN_MTU, IPPROTO_IPV6, OA_RW, OA_RW, OP_NP, 0,
205     sizeof (int), 0 },
206 { IPV6_V6ONLY,  IPPROTO_IPV6, OA_RW, OA_RW, OP_NP, 0,
207     sizeof (int), 0 },
209 /* Enable receipt of ancillary data */
210 { IPV6_RECVPKTINFO, IPPROTO_IPV6, OA_RW, OA_RW, OP_NP, 0,
211     sizeof (int), 0 },
212 { IPV6_RECVHOPLIMIT, IPPROTO_IPV6, OA_RW, OA_RW, OP_NP, 0,
213     sizeof (int), 0 },
214 { IPV6_RECVHOPOPTS, IPPROTO_IPV6, OA_RW, OA_RW, OP_NP, 0,
215     sizeof (int), 0 },
216 { _OLD_IPV6_RECVDSTOPTS, IPPROTO_IPV6, OA_RW, OA_RW, OP_NP, 0,
217     sizeof (int), 0 },
218 { IPV6_RECVDSTOPTS, IPPROTO_IPV6, OA_RW, OA_RW, OP_NP, 0,
219     sizeof (int), 0 },
220 { IPV6_RECVRTHDR,  IPPROTO_IPV6, OA_RW, OA_RW, OP_NP, 0,
221     sizeof (int), 0 },
222 { IPV6_RECVRTHDRDSTOPTS, IPPROTO_IPV6, OA_RW, OA_RW, OP_NP, 0,
223     sizeof (int), 0 },
224 { IPV6_RECVTCLASS,  IPPROTO_IPV6, OA_RW, OA_RW, OP_NP, 0,
225     sizeof (int), 0 },
227 { IPV6_SEC_OPT,   IPPROTO_IPV6, OA_RW, OA_RW, OP_NP, OP_NODEFAULT,
228     sizeof (ipsec_req_t), -1 /* not initialized */ },
229 { IPV6_SRC_PREFERENCES, IPPROTO_IPV6, OA_RW, OA_RW, OP_NP, 0,
230     sizeof (uint32_t), IPV6_PREFER_SRC_DEFAULT },
231 };
233 /*
234  * Table of all supported levels
235  * Note: Some levels (e.g. XTI_GENERIC) may be valid but may not have
236  * any supported options so we need this info separately.
237  *
238  * This is needed only for topmost tpi providers and is used only by
239  * XTI interfaces.
240  */
241 optlevel_t    tcp_valid_levels_arr[] = {
242     XTI_GENERIC,
243     SOL_SOCKET,
244     IPPROTO_TCP,
245     IPPROTO_IP,
246     IPPROTO_IPV6
247 };
250 #define TCP_OPT_ARR_CNT        A_CNT(tcp_opt_arr)
251 #define TCP_VALID_LEVELS_CNT  A_CNT(tcp_valid_levels_arr)
253 uint_t tcp_max_optsize; /* initialized when TCP driver is loaded */
255 /*
256  * Initialize option database object for TCP
257  *
258  * This object represents database of options to search passed to
259  * {sock, tpi}optcom_req() interface routine to take care of option

```

```

260 * management and associated methods.
261 */

263 optdb_obj_t tcp_opt_obj = {
264     tcp_opt_default,      /* TCP default value function pointer */
265     tcp_tpi_opt_get,     /* TCP get function pointer */
266     tcp_tpi_opt_set,     /* TCP set function pointer */
267     TCP_OPT_ARR_CNT,    /* TCP option database count of entries */
268     tcp_opt_arr,        /* TCP option database */
269     TCP_VALID_LEVELS_CNT, /* TCP valid level count of entries */
270     tcp_valid_levels_arr /* TCP valid level array */
271 };

273 /* Maximum TCP initial cwin (start/restart). */
274 #define TCP_MAX_INIT_CWND    16

276 static int tcp_max_init_cwnd = TCP_MAX_INIT_CWND;

278 /*
279  * Some TCP options can be "set" by requesting them in the option
280  * buffer. This is needed for XTI feature test though we do not
281  * allow it in general. We interpret that this mechanism is more
282  * applicable to OSI protocols and need not be allowed in general.
283  * This routine filters out options for which it is not allowed (most)
284  * and lets through those (few) for which it is. [ The XTI interface
285  * test suite specifics will imply that any XTI_GENERIC level XTI_* if
286  * ever implemented will have to be allowed here ].
287  */
288 static boolean_t
289 tcp_allow_connopt_set(int level, int name)
290 {
291     switch (level) {
292     case IPPROTO_TCP:
293         switch (name) {
294             case TCP_NODELAY:
295                 return (B_TRUE);
296             default:
297                 return (B_FALSE);
298         }
299     /*NOTREACHED*/
300     default:
301         return (B_FALSE);
302     }
303     /*NOTREACHED*/
304 }

305 }

307 /*
308  * This routine gets default values of certain options whose default
309  * values are maintained by protocol specific code
310  */
311 /* ARGSUSED */
312 static int
313 tcp_opt_default(queue_t *q, int level, int name, uchar_t *ptr)
314 {
315     int32_t *i1 = (int32_t *)ptr;
316     tcp_stack_t *tcps = Q_TO_TCP(q)->tcp_tcps;

318     switch (level) {
319     case IPPROTO_TCP:
320         switch (name) {
321             case TCP_NOTIFY_THRESHOLD:
322                 *i1 = tcps->tcps_ip_notify_interval;
323                 break;
324             case TCP_ABORT_THRESHOLD:

```

```

325         *i1 = tcps->tcps_ip_abort_interval;
326         break;
327     case TCP_CONN_NOTIFY_THRESHOLD:
328         *i1 = tcps->tcps_ip_notify_cinterval;
329         break;
330     case TCP_CONN_ABORT_THRESHOLD:
331         *i1 = tcps->tcps_ip_abort_cinterval;
332         break;
333     default:
334         return (-1);
335     }
336     break;
337 case IPPROTO_IP:
338     switch (name) {
339     case IP_TTL:
340         *i1 = tcps->tcps_ipv4_ttl;
341         break;
342     default:
343         return (-1);
344     }
345     break;
346 case IPPROTO_IPV6:
347     switch (name) {
348     case IPV6_UNICAST_HOPS:
349         *i1 = tcps->tcps_ipv6_hoplimit;
350         break;
351     default:
352         return (-1);
353     }
354     break;
355 default:
356     return (-1);
357 }
358 return (sizeof (int));
359 }

```

unchanged_portion_omitted

```

*****
32213 Wed Aug  8 12:51:11 2012
new/usr/src/uts/common/inet/tcp/tcp_socket.c
3065 some functions in the tcp module can be static
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
24 */

26 /* This file contains all TCP kernel socket related functions. */

28 #include <sys/types.h>
29 #include <sys/strlog.h>
30 #include <sys/policy.h>
31 #include <sys/sockio.h>
32 #include <sys/strsubr.h>
33 #include <sys/strsun.h>
34 #include <sys/queue_impl.h>
35 #include <sys/queue.h>
36 #define _SUN_TPI_VERSION 2
37 #include <sys/tihdr.h>
38 #include <sys/timod.h>
39 #include <sys/tpicommon.h>
40 #include <sys/socketvar.h>

42 #include <inet/common.h>
43 #include <inet/proto_set.h>
44 #include <inet/ip.h>
45 #include <inet/tcp.h>
46 #include <inet/tcp_impl.h>

48 static void    tcp_activate(sock_lower_handle_t, sock_upper_handle_t,
49                          sock_upcalls_t *, int, cred_t *);
50 static int     tcp_accept(sock_lower_handle_t, sock_lower_handle_t,
51                          sock_upper_handle_t, cred_t *);
52 static int     tcp_bind(sock_lower_handle_t, struct sockaddr *,
53                          socklen_t, cred_t *);
54 static int     tcp_listen(sock_lower_handle_t, int, cred_t *);
55 static int     tcp_connect(sock_lower_handle_t, const struct sockaddr *,
56                          socklen_t, sock_connid_t *, cred_t *);
57 static int     tcp_getpeername(sock_lower_handle_t, struct sockaddr *,
58                          socklen_t *, cred_t *);
59 static int     tcp_getsockname(sock_lower_handle_t, struct sockaddr *,
60                          socklen_t *, cred_t *);
61 #endif /* ! codereview */

```

```

62 static int     tcp_getsockopt(sock_lower_handle_t, int, int, void *,
63                          socklen_t *, cred_t *);
64 static int     tcp_setsockopt(sock_lower_handle_t, int, int, const void *,
65                          socklen_t, cred_t *);
66 static int     tcp_sendmsg(sock_lower_handle_t, mblk_t *, struct nmsg_hdr *,
67                          cred_t *);
68 static int     tcp_shutdown(sock_lower_handle_t, int, cred_t *);
69 static void    tcp_clr_flowctrl(sock_lower_handle_t);
70 static int     tcp_ioctl(sock_lower_handle_t, int, intptr_t, int, int32_t *,
71                          cred_t *);
72 static int     tcp_close(sock_lower_handle_t, int, cred_t *);

74 sock_downcalls_t sock_tcp_downcalls = {
75     tcp_activate,
76     tcp_accept,
77     tcp_bind,
78     tcp_listen,
79     tcp_connect,
80     tcp_getpeername,
81     tcp_getsockname,
82     tcp_getsockopt,
83     tcp_setsockopt,
84     tcp_sendmsg,
85     NULL,
86     NULL,
87     NULL,
88     tcp_shutdown,
89     tcp_clr_flowctrl,
90     tcp_ioctl,
91     tcp_close,
92 };
    _____
    unchanged_portion_omitted

130 /*ARGSUSED*/
131 static int
132 tcp_accept(sock_lower_handle_t lproto_handle,
133            sock_lower_handle_t eproto_handle, sock_upper_handle_t sock_handle,
134            cred_t *cr)
135 {
136     conn_t *lconnp, *econnp;
137     tcp_t *listener, *eager;

139     /* All Solaris components should pass a cred for this operation. */
140     ASSERT(cr != NULL);

142 #endif /* ! codereview */
143 /*
144  * KSSL can move a socket from one listener to another, in which
145  * case 'lproto_handle' points to the new listener. To ensure that
146  * the original listener is used the information is obtained from
147  * the eager.
148  */
149     econnp = (conn_t *)eproto_handle;
150     eager = econnp->conn_tcp;
151     ASSERT(IPCL_IS_NONSTR(econnp));
152     ASSERT(eager->tcp_listener != NULL);
153     listener = eager->tcp_listener;
154     lconnp = (conn_t *)listener->tcp_connp;
155     ASSERT(listener->tcp_state == TCPS_LISTEN);
156     ASSERT(lconnp->conn_upper_handle != NULL);

158 /*
159  * It is possible for the accept thread to race with the thread that
160  * made the su_newconn upcall in tcp_newconn_notify. Both
161  * tcp_newconn_notify and tcp_accept require that conn_upper_handle

```

```

162  * and conn_upcalls be set before returning, so they both write to
163  * them. However, we're guaranteed that the value written is the same
164  * for both threads.
165  */
166  ASSERT(econnp->conn_upper_handle == NULL ||
167         econnp->conn_upper_handle == sock_handle);
168  ASSERT(econnp->conn_upcalls == NULL ||
169         econnp->conn_upcalls == lconnp->conn_upcalls);
170  econnp->conn_upper_handle = sock_handle;
171  econnp->conn_upcalls = lconnp->conn_upcalls;

173  ASSERT(econnp->conn_netstack ==
174         listener->tcp_connp->conn_netstack);
175  ASSERT(eager->tcp_tcps == listener->tcp_tcps);

177  /*
178  * We should have a minimum of 2 references on the conn at this
179  * point. One for TCP and one for the newconn notification
180  * (which is now taken over by IP). In the normal case we would
181  * also have another reference (making a total of 3) for the conn
182  * being in the classifier hash list. However the eager could have
183  * received an RST subsequently and tcp_closei_local could have
184  * removed the eager from the classifier hash list, hence we can't
185  * assert that reference.
186  */
187  ASSERT(econnp->conn_ref >= 2);

189  mutex_enter(&listener->tcp_eager_lock);
190  /*
191  * Non-STREAMS listeners never defer the notification of new
192  * connections.
193  */
194  ASSERT(!listener->tcp_eager_prev_q0->tcp_conn_def_q0);
195  tcp_eager_unlink(eager);
196  mutex_exit(&listener->tcp_eager_lock);
197  CONN_DEC_REF(listener->tcp_connp);

199  return ((eager->tcp_state < TCPS_ESTABLISHED) ? ECONNABORTED : 0);
200 }

202 static int
203 tcp_bind(sock_lower_handle_t proto_handle, struct sockaddr *sa,
204          socklen_t len, cred_t *cr)
205 {
206     int          error;
207     conn_t      *connp = (conn_t *)proto_handle;

209     /* All Solaris components should pass a cred for this operation. */
210     ASSERT(cr != NULL);
211     ASSERT(connp->conn_upper_handle != NULL);

213     error = squeue_synch_enter(connp, NULL);
214     if (error != 0) {
215         /* failed to enter */
216         return (ENOSR);
217     }

219     /* binding to a NULL address really means unbind */
220     if (sa == NULL) {
221         if (connp->conn_tcp->tcp_state < TCPS_LISTEN)
222             error = tcp_do_unbind(connp);
223         else
224             error = EINVAL;
225     } else {
226         error = tcp_do_bind(connp, sa, len, cr, B_TRUE);
227     }

```

```

229     squeue_synch_exit(connp);

231     if (error < 0) {
232         if (error == -TOUTSTATE)
233             error = EINVAL;
234         else
235             error = proto_tltosyserr(-error);
236     }

238     return (error);
239 }

241 /* ARGSUSED */
242 static int
243 tcp_listen(sock_lower_handle_t proto_handle, int backlog, cred_t *cr)
244 {
245     conn_t      *connp = (conn_t *)proto_handle;
246     tcp_t      *tcp = connp->conn_tcp;
247     int          error;

249     ASSERT(connp->conn_upper_handle != NULL);

251     /* All Solaris components should pass a cred for this operation. */
252     ASSERT(cr != NULL);

254     error = squeue_synch_enter(connp, NULL);
255     if (error != 0) {
256         /* failed to enter */
257         return (ENOBUFS);
258     }

260     error = tcp_do_listen(connp, NULL, 0, backlog, cr, B_FALSE);
261     if (error == 0) {
262         /*
263          * sockfs needs to know what's the maximum number of socket
264          * that can be queued on the listener.
265          */
266         (*connp->conn_upcalls->su_opctl)(connp->conn_upper_handle,
267                                       SOCK_OPCTL_ENAB_ACCEPT,
268                                       (uintptr_t)(tcp->tcp_conn_req_max +
269                                                  tcp->tcp_tcps->tcps_conn_req_max_q0));
270     } else if (error < 0) {
271         if (error == -TOUTSTATE)
272             error = EINVAL;
273         else
274             error = proto_tltosyserr(-error);
275     }
276     squeue_synch_exit(connp);
277     return (error);
278 }

280 static int
281 tcp_connect(sock_lower_handle_t proto_handle, const struct sockaddr *sa,
282            socklen_t len, sock_connid_t *id, cred_t *cr)
283 {
284     conn_t      *connp = (conn_t *)proto_handle;
285     int          error;

287     ASSERT(connp->conn_upper_handle != NULL);

289     /* All Solaris components should pass a cred for this operation. */
290     ASSERT(cr != NULL);

292     error = proto_verify_ip_addr(connp->conn_family, sa, len);
293     if (error != 0) {

```

```

294         return (error);
295     }

297     error = queue_synch_enter(connp, NULL);
298     if (error != 0) {
299         /* failed to enter */
300         return (ENOSR);
301     }

303     /*
304      * TCP supports quick connect, so no need to do an implicit bind
305      */
306     error = tcp_do_connect(connp, sa, len, cr, curproc->p_pid);
307     if (error == 0) {
308         *id = connp->conn_tcp->tcp_connid;
309     } else if (error < 0) {
310         if (error == -TOUTSTATE) {
311             switch (connp->conn_tcp->tcp_state) {
312                 case TCPS_SYN_SENT:
313                     error = EALREADY;
314                     break;
315                 case TCPS_ESTABLISHED:
316                     error = EISCONN;
317                     break;
318                 case TCPS_LISTEN:
319                     error = EOPNOTSUPP;
320                     break;
321                 default:
322                     error = EINVAL;
323                     break;
324             }
325         } else {
326             error = proto_tltosyserr(-error);
327         }
328     }

330     if (connp->conn_tcp->tcp_loopback) {
331         struct sock_proto_props sopp;

333         sopp.sopp_flags = SOCKOPT_LOOPBACK;
334         sopp.sopp_loopback = B_TRUE;

336         (*connp->conn_upcalls->su_set_proto_props)(
337             connp->conn_upper_handle, &sopp);
338     }
339 done:
340     queue_synch_exit(connp);

342     return ((error == 0) ? EINPROGRESS : error);
343 }

345 /* ARGSUSED3 */
346 static int
347 int
348 tcp_getpeername(sock_lower_handle_t proto_handle, struct sockaddr *addr,
349                 socklen_t *addrlenp, cred_t *cr)
350 {
351     conn_t *connp = (conn_t *)proto_handle;
352     tcp_t *tcp = connp->conn_tcp;

353     /* All Solaris components should pass a cred for this operation. */
354     ASSERT(cr != NULL);

356     ASSERT(tcp != NULL);
357     if (tcp->tcp_state < TCPS_SYN_RCVD)
358         return (ENOTCONN);

```

```

360         return (conn_getpeername(connp, addr, addrlenp));
361     }

363 /* ARGSUSED3 */
364 static int
365 int
366 tcp_getsockname(sock_lower_handle_t proto_handle, struct sockaddr *addr,
367                 socklen_t *addrlenp, cred_t *cr)
368 {
369     conn_t *connp = (conn_t *)proto_handle;

370     /* All Solaris components should pass a cred for this operation. */
371     ASSERT(cr != NULL);

373     return (conn_getsockname(connp, addr, addrlenp));
374 }

376 /* returns UNIX error, the optlen is a value-result arg */
377 static int
378 tcp_getsockopt(sock_lower_handle_t proto_handle, int level, int option_name,
379               void *optvalp, socklen_t *optlen, cred_t *cr)
380 {
381     conn_t *connp = (conn_t *)proto_handle;
382     int error;
383     t_uscalar_t max_optbuf_len;
384     void *optvalp_buf;
385     int len;

387     ASSERT(connp->conn_upper_handle != NULL);

389     /* All Solaris components should pass a cred for this operation. */
390     ASSERT(cr != NULL);

392 #endif /* ! codereview */
393     error = proto_opt_check(level, option_name, *optlen, &max_optbuf_len,
394                             tcp_opt_obj.odt_opt_des_arr,
395                             tcp_opt_obj.odt_opt_arr_cnt,
396                             B_FALSE, B_TRUE, cr);
397     if (error != 0) {
398         if (error < 0) {
399             error = proto_tltosyserr(-error);
400         }
401         return (error);
402     }

404     optvalp_buf = kmem_alloc(max_optbuf_len, KM_SLEEP);

406     error = queue_synch_enter(connp, NULL);
407     if (error == ENOMEM) {
408         kmem_free(optvalp_buf, max_optbuf_len);
409         return (ENOMEM);
410     }

412     len = tcp_opt_get(connp, level, option_name, optvalp_buf);
413     queue_synch_exit(connp);

415     if (len == -1) {
416         kmem_free(optvalp_buf, max_optbuf_len);
417         return (EINVAL);
418     }

420     /*
421      * update optlen and copy option value
422      */
423     t_uscalar_t size = MIN(len, *optlen);

```



```

556         CONN_INC_REF(connp);
558         if (msg->msg_flags & MSG_OOB) {
559             SQUEUE_ENTER_ONE(connp->conn_sq, mp, tcp_output_urgent,
560                 connp, NULL, tcp_queue_flag, SQTAG_TCP_OUTPUT);
561         } else {
562             SQUEUE_ENTER_ONE(connp->conn_sq, mp, tcp_output,
563                 connp, NULL, tcp_queue_flag, SQTAG_TCP_OUTPUT);
564         }
566         return (0);
568     default:
569         ASSERT(0);
570     }
572     freemsg(mp);
573     return (0);
574 }
576 /* ARGSUSED */
577 static int
578 tcp_shutdown(sock_lower_handle_t proto_handle, int how, cred_t *cr)
579 {
580     conn_t *connp = (conn_t *)proto_handle;
581     tcp_t *tcp = connp->conn_tcp;
583     ASSERT(connp->conn_upper_handle != NULL);
585     /* All Solaris components should pass a cred for this operation. */
586     ASSERT(cr != NULL);
588     /*
589      * X/Open requires that we check the connected state.
590      */
591     if (tcp->tcp_state < TCPS_SYN_SENT)
592         return (ENOTCONN);
594     /* shutdown the send side */
595     if (how != SHUT_RD) {
596         mblk_t *bp;
598         bp = allocb_wait(0, BPRI_HI, STR_NOSIG, NULL);
599         CONN_INC_REF(connp);
600         SQUEUE_ENTER_ONE(connp->conn_sq, bp, tcp_shutdown_output,
601             connp, NULL, SQ_NODRAIN, SQTAG_TCP_SHUTDOWN_OUTPUT);
603         (*connp->conn_upcalls->su_opctl)(connp->conn_upper_handle,
604             SOCK_OPCTL_SHUT_SEND, 0);
605     }
607     /* shutdown the rcv side */
608     if (how != SHUT_WR)
609         (*connp->conn_upcalls->su_opctl)(connp->conn_upper_handle,
610             SOCK_OPCTL_SHUT_RECV, 0);
612     return (0);
613 }
615 static void
616 tcp_clr_flowctrl(sock_lower_handle_t proto_handle)
617 {
618     conn_t *connp = (conn_t *)proto_handle;
619     tcp_t *tcp = connp->conn_tcp;
620     mblk_t *mp;
621     int error;

```

```

623     ASSERT(connp->conn_upper_handle != NULL);
625     /*
626      * If tcp->tcp_rsrv_mp == NULL, it means that tcp_clr_flowctrl()
627      * is currently running.
628      */
629     mutex_enter(&tcp->tcp_rsrv_mp_lock);
630     if ((mp = tcp->tcp_rsrv_mp) == NULL) {
631         mutex_exit(&tcp->tcp_rsrv_mp_lock);
632         return;
633     }
634     tcp->tcp_rsrv_mp = NULL;
635     mutex_exit(&tcp->tcp_rsrv_mp_lock);
637     error = squeue_synch_enter(connp, mp);
638     ASSERT(error == 0);
640     mutex_enter(&tcp->tcp_rsrv_mp_lock);
641     tcp->tcp_rsrv_mp = mp;
642     mutex_exit(&tcp->tcp_rsrv_mp_lock);
644     if (tcp->tcp_fused) {
645         tcp_fuse_backenable(tcp);
646     } else {
647         tcp->tcp_rwnd = connp->conn_rcvbuf;
648         /*
649          * Send back a window update immediately if TCP is above
650          * ESTABLISHED state and the increase of the rcv window
651          * that the other side knows is at least 1 MSS after flow
652          * control is lifted.
653          */
654         if (tcp->tcp_state >= TCPS_ESTABLISHED &&
655             tcp_rwnd_reopen(tcp) == TH_ACK_NEEDED) {
656             tcp_xmit_ctl(NULL, tcp,
657                 (tcp->tcp_swnd == 0) ? tcp->tcp_suna :
658                 tcp->tcp_snxt, tcp->tcp_rnxt, TH_ACK);
659         }
660     }
662     squeue_synch_exit(connp);
663 }
665 /* ARGSUSED */
666 static int
667 tcp_ioctl(sock_lower_handle_t proto_handle, int cmd, intptr_t arg,
668     int mode, int32_t *rvalp, cred_t *cr)
669 {
670     conn_t *connp = (conn_t *)proto_handle;
671     int error;
673     ASSERT(connp->conn_upper_handle != NULL);
675     /* All Solaris components should pass a cred for this operation. */
676     ASSERT(cr != NULL);
678     /*
679      * If we don't have a helper stream then create one.
680      * ip_create_helper_stream takes care of locking the conn_t,
681      * so this check for NULL is just a performance optimization.
682      */
683     if (connp->conn_helper_info == NULL) {
684         tcp_stack_t *tcps = connp->conn_tcp->tcp_tcps;
686         /*
687          * Create a helper stream for non-STREAMS socket.

```



```

688     */
689     error = ip_create_helper_stream(connp, tcps->tcps_ldi_ident);
690     if (error != 0) {
691         ip0dbg(("tcp_ioctl: create of IP helper stream "
692             "failed %d\n", error));
693         return (error);
694     }
695 }

697 switch (cmd) {
698     case ND_SET:
699     case ND_GET:
700     case _SIOCSOCKFALLBACK:
701     case TCP_IOC_ABORT_CONN:
702     case TI_GETPEERNAME:
703     case TI_GETMYNAME:
704         ipldbg(("tcp_ioctl: cmd 0x%x on non streams socket",
705             cmd));
706         error = EINVAL;
707         break;
708     default:
709         /*
710          * If the conn is not closing, pass on to IP using
711          * helper stream. Bump the ioctlref to prevent tcp_close
712          * from closing the rq/wq out from underneath the ioctl
713          * if it ends up queued or aborted/interrupted.
714          */
715         mutex_enter(&connp->conn_lock);
716         if (connp->conn_state_flags & (CONN_CLOSING)) {
717             mutex_exit(&connp->conn_lock);
718             error = EINVAL;
719             break;
720         }
721         CONN_INC_IOCTLREF_LOCKED(connp);
722         error = ldi_ioctl(connp->conn_helper_info->iphs_handle,
723             cmd, arg, mode, cr, rvalp);
724         CONN_DEC_IOCTLREF(connp);
725         break;
726     }
727     return (error);
728 }

730 /* ARGSUSED */
731 static int
732 tcp_close(sock_lower_handle_t proto_handle, int flags, cred_t *cr)
733 {
734     conn_t *connp = (conn_t *)proto_handle;
735
736     ASSERT(connp->conn_upper_handle != NULL);

738     /* All Solaris components should pass a cred for this operation. */
739     ASSERT(cr != NULL);

741     tcp_close_common(connp, flags);

743     ip_free_helper_stream(connp);

745     /*
746      * Drop IP's reference on the conn. This is the last reference
747      * on the connp if the state was less than established. If the
748      * connection has gone into timewait state, then we will have
749      * one ref for the TCP and one more ref (total of two) for the
750      * classifier connected hash list (a timewait connections stays
751      * in connected hash till closed).
752      *
753      * We can't assert the references because there might be other

```

```

754     * transient reference places because of some walkers or queued
755     * packets in queue for the timewait state.
756     */
757     CONN_DEC_REF(connp);

759     /*
760     * EINPROGRESS tells sockfs to wait for a 'closed' upcall before
761     * freeing the socket.
762     */
763     return (EINPROGRESS);
764 }

766 /* ARGSUSED */
767 sock_lower_handle_t
768 tcp_create(int family, int type, int proto, sock_downcalls_t **sock_downcalls,
769     uint_t *smodep, int *errorp, int flags, cred_t *credp)
770 {
771     conn_t      *connp;
772     boolean_t   isv6 = family == AF_INET6;

774 #endif /* ! codereview */
775     if (type != SOCK_STREAM || (family != AF_INET && family != AF_INET6) ||
776         (proto != 0 && proto != IPPROTO_TCP)) {
777         *errorp = EPROTONOSUPPORT;
778         return (NULL);
779     }

781     connp = tcp_create_common(credp, isv6, B_TRUE, errorp);
782     if (connp == NULL) {
783         return (NULL);
784     }

786     /*
787     * Put the ref for TCP. Ref for IP was already put
788     * by ipcl_conn_create. Also make the conn_t globally
789     * by ipcl_conn_create. Also Make the conn_t globally
790     * visible to walkers
791     */
792     mutex_enter(&connp->conn_lock);
793     CONN_INC_REF_LOCKED(connp);
794     ASSERT(connp->conn_ref == 2);
795     connp->conn_state_flags &= ~CONN_INCIPIENT;

796     connp->conn_flags |= IPCL_NONSTR;
797     mutex_exit(&connp->conn_lock);

799     ASSERT(errorp != NULL);
800     *errorp = 0;
801     *sock_downcalls = &sock_tcp_downcalls;
802     *smodep = SM_CONNRQUIRED | SM_EXDATA | SM_ACCEPTSUPP |
803         SM_SENDFILESUPP;

805     return ((sock_lower_handle_t)connp);
806 }

    unchanged_portion_omitted

```

```

*****
33177 Wed Aug  8 12:51:11 2012
new/usr/src/uts/common/inet/tcp/tcp_stats.c
3065 some functions in the tcp module can be static
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
24 */

26 #include <sys/types.h>
27 #include <sys/tihdr.h>
28 #include <sys/policy.h>
29 #include <sys/tsol/tnet.h>

31 #include <inet/common.h>
32 #include <inet/ip.h>
33 #include <inet/tcp.h>
34 #include <inet/tcp_impl.h>
35 #include <inet/tcp_stats.h>
36 #include <inet/kstatcom.h>
37 #include <inet/snmpcom.h>

39 static int      tcp_snmp_state(tcp_t *);
40 static int      tcp_kstat_update(kstat_t *, int);
41 static int      tcp_kstat2_update(kstat_t *, int);
39 static int      tcp_kstat_update(kstat_t *kp, int rw);
40 static int      tcp_kstat2_update(kstat_t *kp, int rw);
42 static void      tcp_sum_mib(tcp_stack_t *, mib2_tcp_t *);

44 static void      tcp_add_mib(mib2_tcp_t *, mib2_tcp_t *);
45 static void      tcp_add_stats(tcp_stat_counter_t *, tcp_stat_t *);
46 static void      tcp_clr_stats(tcp_stat_t *);

48 tcp_g_stat_t      tcp_g_statistics;
49 kstat_t           *tcp_g_kstat;

51 /* Translate TCP state to MIB2 TCP state. */
52 static int
53 tcp_snmp_state(tcp_t *tcp)
54 {
55     if (tcp == NULL)
56         return (0);

58     switch (tcp->tcp_state) {
59         case TCPS_CLOSED:

```

```

60     case TCPS_IDLE: /* RFC1213 doesn't have analogue for IDLE & BOUND */
61     case TCPS_BOUND:
62         return (MIB2_TCP_closed);
63     case TCPS_LISTEN:
64         return (MIB2_TCP_listen);
65     case TCPS_SYN_SENT:
66         return (MIB2_TCP_synSent);
67     case TCPS_SYN_RCVD:
68         return (MIB2_TCP_synReceived);
69     case TCPS_ESTABLISHED:
70         return (MIB2_TCP_established);
71     case TCPS_CLOSE_WAIT:
72         return (MIB2_TCP_closeWait);
73     case TCPS_FIN_WAIT_1:
74         return (MIB2_TCP_finWait1);
75     case TCPS_CLOSING:
76         return (MIB2_TCP_closing);
77     case TCPS_LAST_ACK:
78         return (MIB2_TCP_lastAck);
79     case TCPS_FIN_WAIT_2:
80         return (MIB2_TCP_finWait2);
81     case TCPS_TIME_WAIT:
82         return (MIB2_TCP_timeWait);
83     default:
84         return (0);
85     }
86 }
unchanged_portion_omitted

801 /*
802  * To add stats from one mib2_tcp_t to another.  Static fields are not added.
803  * The caller should set them up properly.
804  */
805 static void
806 void
807 tcp_add_mib(mib2_tcp_t *from, mib2_tcp_t *to)
808 {
809     to->tcpActiveOpens += from->tcpActiveOpens;
810     to->tcpPassiveOpens += from->tcpPassiveOpens;
811     to->tcpAttemptFails += from->tcpAttemptFails;
812     to->tcpEstabResets += from->tcpEstabResets;
813     to->tcpInSegs += from->tcpInSegs;
814     to->tcpOutSegs += from->tcpOutSegs;
815     to->tcpRetransSegs += from->tcpRetransSegs;
816     to->tcpOutRsts += from->tcpOutRsts;

817     to->tcpOutDataSegs += from->tcpOutDataSegs;
818     to->tcpOutDataBytes += from->tcpOutDataBytes;
819     to->tcpRetransBytes += from->tcpRetransBytes;
820     to->tcpOutAck += from->tcpOutAck;
821     to->tcpOutAckDelayed += from->tcpOutAckDelayed;
822     to->tcpOutUrg += from->tcpOutUrg;
823     to->tcpOutWinUpdate += from->tcpOutWinUpdate;
824     to->tcpOutWinProbe += from->tcpOutWinProbe;
825     to->tcpOutControl += from->tcpOutControl;
826     to->tcpOutFastRetrans += from->tcpOutFastRetrans;

828     to->tcpInAckBytes += from->tcpInAckBytes;
829     to->tcpInDupAck += from->tcpInDupAck;
830     to->tcpInAckUnsent += from->tcpInAckUnsent;
831     to->tcpInDataInorderSegs += from->tcpInDataInorderSegs;
832     to->tcpInDataInorderBytes += from->tcpInDataInorderBytes;
833     to->tcpInDataUnorderSegs += from->tcpInDataUnorderSegs;
834     to->tcpInDataUnorderBytes += from->tcpInDataUnorderBytes;
835     to->tcpInDataDupSegs += from->tcpInDataDupSegs;
836     to->tcpInDataDupBytes += from->tcpInDataDupBytes;

```

```
837     to->tcpInDataPartDupSegs += from->tcpInDataPartDupSegs;
838     to->tcpInDataPartDupBytes += from->tcpInDataPartDupBytes;
839     to->tcpInDataPastWinSegs += from->tcpInDataPastWinSegs;
840     to->tcpInDataPastWinBytes += from->tcpInDataPastWinBytes;
841     to->tcpInWinProbe += from->tcpInWinProbe;
842     to->tcpInWinUpdate += from->tcpInWinUpdate;
843     to->tcpInClosed += from->tcpInClosed;

845     to->tcpRttNoUpdate += from->tcpRttNoUpdate;
846     to->tcpRttUpdate += from->tcpRttUpdate;
847     to->tcpTimRetrans += from->tcpTimRetrans;
848     to->tcpTimRetransDrop += from->tcpTimRetransDrop;
849     to->tcpTimKeepalive += from->tcpTimKeepalive;
850     to->tcpTimKeepaliveProbe += from->tcpTimKeepaliveProbe;
851     to->tcpTimKeepaliveDrop += from->tcpTimKeepaliveDrop;
852     to->tcpListenDrop += from->tcpListenDrop;
853     to->tcpListenDropQ0 += from->tcpListenDropQ0;
854     to->tcpHalfOpenDrop += from->tcpHalfOpenDrop;
855     to->tcpOutSackRetransSegs += from->tcpOutSackRetransSegs;
856     to->tcpHCInSegs += from->tcpHCInSegs;
857     to->tcpHCOutSegs += from->tcpHCOutSegs;
858 }
```

unchanged_portion_omitted

```

*****
28387 Wed Aug 8 12:51:12 2012
new/usr/src/uts/common/inet/tcp_impl.h
3065 some functions in the tcp module can be static
*****
_____unchanged_portion_omitted_____

339 /* Increment and decrement the number of connections in tcp_stack_t. */
340 #define TCPS_CONN_INC(tcps) \
341     atomic_inc_64( \
342         (uint64_t *)&(tcps)->tcps_sc[CPU->cpu_seqid]->tcp_sc_conn_cnt)

344 #define TCPS_CONN_DEC(tcps) \
345     atomic_dec_64( \
346         (uint64_t *)&(tcps)->tcps_sc[CPU->cpu_seqid]->tcp_sc_conn_cnt)

348 /*
349  * When the system is under memory pressure, stack variable tcps_reclaim is
350  * true, we shorten the connection timeout abort interval to tcp_early_abort
351  * seconds. Defined in tcp.c.
352  */
353 extern uint32_t tcp_early_abort;

355 /*
356  * To reach to an eager in Q0 which can be dropped due to an incoming
357  * new SYN request when Q0 is full, a new doubly linked list is
358  * introduced. This list allows to select an eager from Q0 in O(1) time.
359  * This is needed to avoid spending too much time walking through the
360  * long list of eagers in Q0 when tcp_drop_q0() is called. Each member of
361  * this new list has to be a member of Q0.
362  * This list is headed by listener's tcp_t. When the list is empty,
363  * both the pointers - tcp_eager_next_drop_q0 and tcp_eager_prev_drop_q0,
364  * of listener's tcp_t point to listener's tcp_t itself.
365  *
366  * Given an eager in Q0 and a listener, MAKE_DROPPABLE() puts the eager
367  * in the list. MAKE_UNDROPPABLE() takes the eager out of the list.
368  * These macros do not affect the eager's membership to Q0.
369  */
370 #define MAKE_DROPPABLE(listener, eager) \
371     if ((eager)->tcp_eager_next_drop_q0 == NULL) { \
372         (listener)->tcp_eager_next_drop_q0->tcp_eager_prev_drop_q0 \
373         = (eager); \
374         (eager)->tcp_eager_prev_drop_q0 = (listener); \
375         (eager)->tcp_eager_next_drop_q0 = \
376         (listener)->tcp_eager_next_drop_q0; \
377         (listener)->tcp_eager_next_drop_q0 = (eager); \
378     }

380 #define MAKE_UNDROPPABLE(eager) \
381     if ((eager)->tcp_eager_next_drop_q0 != NULL) { \
382         (eager)->tcp_eager_next_drop_q0->tcp_eager_prev_drop_q0 \
383         = (eager)->tcp_eager_prev_drop_q0; \
384         (eager)->tcp_eager_prev_drop_q0->tcp_eager_next_drop_q0 \
385         = (eager)->tcp_eager_next_drop_q0; \
386         (eager)->tcp_eager_prev_drop_q0 = NULL; \
387         (eager)->tcp_eager_next_drop_q0 = NULL; \
388     }

390 /*
391  * The format argument to pass to tcp_display().
392  * DISP_PORT_ONLY means that the returned string has only port info.
393  * DISP_ADDR_AND_PORT means that the returned string also contains the
394  * remote and local IP address.
395  */
396 #define DISP_PORT_ONLY 1
397 #define DISP_ADDR_AND_PORT 2

```

```

399 #define IP_ADDR_CACHE_SIZE 2048
400 #define IP_ADDR_CACHE_HASH(faddr) \
401     (ntohl(faddr) & (IP_ADDR_CACHE_SIZE -1))

403 /* TCP cwnd burst factor. */
404 #define TCP_CWND_INFINITE 65535
405 #define TCP_CWND_SS 3
406 #define TCP_CWND_NORMAL 5

408 /*
409  * TCP reassembly macros. We hide starting and ending sequence numbers in
410  * b_next and b_prev of messages on the reassembly queue. The messages are
411  * chained using b_cont. These macros are used in tcp_reass() so we don't
412  * have to see the ugly casts and assignments.
413  */
414 #define TCP_REASS_SEQ(mp) ((uint32_t)(uintptr_t)((mp)->b_next))
415 #define TCP_REASS_SET_SEQ(mp, u) ((mp)->b_next = \
416     (mbk_t *) (uintptr_t)(u))
417 #define TCP_REASS_END(mp) ((uint32_t)(uintptr_t)((mp)->b_prev))
418 #define TCP_REASS_SET_END(mp, u) ((mp)->b_prev = \
419     (mbk_t *) (uintptr_t)(u))

421 #define tcps_time_wait_interval tcps_propinfo_tbl[0].prop_cur_uval
422 #define tcps_conn_req_max_q tcps_propinfo_tbl[1].prop_cur_uval
423 #define tcps_conn_req_max_q0 tcps_propinfo_tbl[2].prop_cur_uval
424 #define tcps_conn_req_min tcps_propinfo_tbl[3].prop_cur_uval
425 #define tcps_conn_grace_period tcps_propinfo_tbl[4].prop_cur_uval
426 #define tcps_cwnd_max tcps_propinfo_tbl[5].prop_cur_uval
427 #define tcps_dbg tcps_propinfo_tbl[6].prop_cur_uval
428 #define tcps_smallest_nonpriv_port tcps_propinfo_tbl[7].prop_cur_uval
429 #define tcps_ip_abort_cinterval tcps_propinfo_tbl[8].prop_cur_uval
430 #define tcps_ip_abort_linterval tcps_propinfo_tbl[9].prop_cur_uval
431 #define tcps_ip_abort_interval tcps_propinfo_tbl[10].prop_cur_uval
432 #define tcps_ip_notify_cinterval tcps_propinfo_tbl[11].prop_cur_uval
433 #define tcps_ip_notify_interval tcps_propinfo_tbl[12].prop_cur_uval
434 #define tcps_ipv4_ttl tcps_propinfo_tbl[13].prop_cur_uval
435 #define tcps_keepalive_interval_high tcps_propinfo_tbl[14].prop_cur_uval
436 #define tcps_keepalive_interval tcps_propinfo_tbl[14].prop_cur_uval
437 #define tcps_keepalive_interval_low tcps_propinfo_tbl[14].prop_min_uval
438 #define tcps_maxpsz_multiplier tcps_propinfo_tbl[15].prop_cur_uval
439 #define tcps_mss_def_ipv4 tcps_propinfo_tbl[16].prop_cur_uval
440 #define tcps_mss_max_ipv4 tcps_propinfo_tbl[17].prop_cur_uval
441 #define tcps_mss_min tcps_propinfo_tbl[18].prop_cur_uval
442 #define tcps_naglim_def tcps_propinfo_tbl[19].prop_cur_uval
443 #define tcps_rexmit_interval_initial_high tcps_propinfo_tbl[20].prop_max_uval
444 #define tcps_rexmit_interval_initial tcps_propinfo_tbl[20].prop_cur_uval
445 #define tcps_rexmit_interval_initial_low tcps_propinfo_tbl[20].prop_min_uval
446 #define tcps_rexmit_interval_max_high tcps_propinfo_tbl[21].prop_max_uval
447 #define tcps_rexmit_interval_max tcps_propinfo_tbl[21].prop_cur_uval
448 #define tcps_rexmit_interval_max_low tcps_propinfo_tbl[21].prop_min_uval
449 #define tcps_rexmit_interval_min_high tcps_propinfo_tbl[22].prop_max_uval
450 #define tcps_rexmit_interval_min tcps_propinfo_tbl[22].prop_cur_uval
451 #define tcps_rexmit_interval_min_low tcps_propinfo_tbl[22].prop_min_uval
452 #define tcps_deferred_ack_interval tcps_propinfo_tbl[23].prop_cur_uval
453 #define tcps_snd_lowat_fraction tcps_propinfo_tbl[24].prop_cur_uval
454 #define tcps_dupack_fast_retransmit tcps_propinfo_tbl[25].prop_cur_uval
455 #define tcps_ignore_path_mtu tcps_propinfo_tbl[26].prop_cur_bval
456 #define tcps_smallest_anon_port tcps_propinfo_tbl[27].prop_cur_uval
457 #define tcps_largest_anon_port tcps_propinfo_tbl[28].prop_cur_uval
458 #define tcps_xmit_hiwat tcps_propinfo_tbl[29].prop_cur_uval
459 #define tcps_xmit_lowat tcps_propinfo_tbl[30].prop_cur_uval
460 #define tcps_recv_hiwat tcps_propinfo_tbl[31].prop_cur_uval
461 #define tcps_recv_hiwat_minmss tcps_propinfo_tbl[32].prop_cur_uval

```

```

464 #define tcps_fin_wait_2_flush_interval_high \
465         tcps_propinfo_tbl[33].prop_max_uval
466 #define tcps_fin_wait_2_flush_interval_low \
467         tcps_propinfo_tbl[33].prop_cur_uval
468 #define tcps_fin_wait_2_flush_interval_min \
469         tcps_propinfo_tbl[33].prop_min_uval
470 #define tcps_max_buf \
471         tcps_propinfo_tbl[34].prop_cur_uval
472 #define tcps_strong_iss \
473         tcps_propinfo_tbl[35].prop_cur_uval
474 #define tcps_rtt_updates \
475         tcps_propinfo_tbl[36].prop_cur_uval
476 #define tcps_wscales_always \
477         tcps_propinfo_tbl[37].prop_cur_bval
478 #define tcps_tstamp_always \
479         tcps_propinfo_tbl[38].prop_cur_bval
480 #define tcps_tstamp_if_wscales \
481         tcps_propinfo_tbl[39].prop_cur_bval
482 #define tcps_rexmit_interval_extra \
483         tcps_propinfo_tbl[40].prop_cur_uval
484 #define tcps_deferred_acks_max \
485         tcps_propinfo_tbl[41].prop_cur_uval
486 #define tcps_slow_start_after_idle \
487         tcps_propinfo_tbl[42].prop_cur_uval
488 #define tcps_slow_start_initial \
489         tcps_propinfo_tbl[43].prop_cur_uval
490 #define tcps_sack_permitted \
491         tcps_propinfo_tbl[44].prop_cur_uval
492 #define tcps_ipv6_hoplimit \
493         tcps_propinfo_tbl[45].prop_cur_uval
494 #define tcps_mss_def_ipv6 \
495         tcps_propinfo_tbl[46].prop_cur_uval
496 #define tcps_mss_max_ipv6 \
497         tcps_propinfo_tbl[47].prop_cur_uval
498 #define tcps_rev_src_routes \
499         tcps_propinfo_tbl[48].prop_cur_bval
500 #define tcps_local_dack_interval \
501         tcps_propinfo_tbl[49].prop_cur_uval
502 #define tcps_local_dacks_max \
503         tcps_propinfo_tbl[50].prop_cur_uval
504 #define tcps_ecn_permitted \
505         tcps_propinfo_tbl[51].prop_cur_uval
506 #define tcps_rst_sent_rate_enabled \
507         tcps_propinfo_tbl[52].prop_cur_bval
508 #define tcps_rst_sent_rate \
509         tcps_propinfo_tbl[53].prop_cur_uval
510 #define tcps_push_timer_interval \
511         tcps_propinfo_tbl[54].prop_cur_uval
512 #define tcps_use_smss_as_mss_opt \
513         tcps_propinfo_tbl[55].prop_cur_bval
514 #define tcps_keepalive_abort_interval_high \
515         tcps_propinfo_tbl[56].prop_max_uval
516 #define tcps_keepalive_abort_interval_low \
517         tcps_propinfo_tbl[56].prop_min_uval
518 #define tcps_keepalive_abort_interval_cur \
519         tcps_propinfo_tbl[56].prop_cur_uval
520 #define tcps_wroff_xtra \
521         tcps_propinfo_tbl[57].prop_cur_uval
522 #define tcps_dev_flow_ctl \
523         tcps_propinfo_tbl[58].prop_cur_bval
524 #define tcps_reass_timeout \
525         tcps_propinfo_tbl[59].prop_cur_uval
526 #define tcps_iss_incr \
527         tcps_propinfo_tbl[65].prop_cur_uval

```

```

502 extern struct qinit tcp_rinitv4, tcp_rinitv6;
503 extern boolean_t do_tcp_fusion;

505 /*
506  * Object to represent database of options to search passed to
507  * {sock,tpi}optcom_req() interface routine to take care of option
508  * management and associated methods.
509  */
510 extern optdb_obj_t      tcp_opt_obj;
511 extern uint_t          tcp_max_optsize;

513 extern int tcp_queue_flag;

515 extern uint_t tcp_free_list_max_cnt;

517 /*
518  * Functions in tcp.c.
519  */
520 extern void tcp_acceptor_hash_insert(t_uscalar_t, tcp_t *);
521 extern tcp_t *tcp_acceptor_hash_lookup(t_uscalar_t, tcp_stack_t *);
522 extern void tcp_acceptor_hash_remove(tcp_t *);
523 extern mblk_t *tcp_ack_mp(tcp_t *);
524 extern int tcp_build_hdrs(tcp_t *);
525 extern void tcp_cleanup(tcp_t *);
526 extern int tcp_clean_death(tcp_t *, int);
527 extern void tcp_clean_death_wrapper(void *, mblk_t *, void *,
528         ip_rcv_attr_t *);
529 extern void tcp_close_common(conn_t *, int);

```

```

530 extern void tcp_close_detached(tcp_t *);
531 extern void tcp_close_mpp(mblk_t **);
532 extern void tcp_closei_local(tcp_t *);
533 extern sock_lower_handle_t tcp_create(int, int, int, sock_downcalls_t **,
534         uint_t *, int *, int, cred_t *);
535 extern conn_t *tcp_create_common(cred_t *, boolean_t, boolean_t, int *);
536 extern void tcp_disconnect(tcp_t *, mblk_t *);
537 extern char *tcp_display(tcp_t *, char *, char);
538 extern int tcp_do_bind(conn_t *, struct sockaddr *, socklen_t, cred_t *,
539         boolean_t);
540 extern int tcp_do_connect(conn_t *, const struct sockaddr *, socklen_t,
541         cred_t *, pid_t);
542 extern int tcp_do_listen(conn_t *, struct sockaddr *, socklen_t, int,
543         cred_t *, boolean_t);
544 extern int tcp_do_unbind(conn_t *);
545 extern boolean_t tcp_eager_blowoff(tcp_t *, t_scalar_t);
546 extern void tcp_eager_cleanup(tcp_t *, boolean_t);
547 extern void tcp_eager_kill(void *, mblk_t *, void *, ip_rcv_attr_t *);
548 extern void tcp_eager_unlink(tcp_t *);
549 extern int tcp_getpeername(sock_lower_handle_t, struct sockaddr *,
550         socklen_t *, cred_t *);
551 extern int tcp_getsockname(sock_lower_handle_t, struct sockaddr *,
552         socklen_t *, cred_t *);
553 extern void tcp_init_values(tcp_t *, tcp_t *);
554 extern void tcp_ipsec_cleanup(tcp_t *);
555 extern int tcp_maxpsz_set(tcp_t *, boolean_t);
556 extern void tcp_mss_set(tcp_t *, uint32_t);
557 extern void tcp_reinput(conn_t *, mblk_t *, ip_rcv_attr_t *, ip_stack_t *);
558 extern void tcp_rsrv(queue_t *);
559 extern uint_t tcp_rwnd_reopen(tcp_t *);
560 extern int tcp_rwnd_set(tcp_t *, uint32_t);
561 extern int tcp_set_destination(tcp_t *);
562 extern void tcp_set_ws_value(tcp_t *);
563 extern void tcp_stop_lingering(tcp_t *);
564 extern void tcp_update_pmtu(tcp_t *, boolean_t);
565 extern mblk_t *tcp_zcopy_backoff(tcp_t *, mblk_t *, boolean_t);
566 extern boolean_t tcp_zcopy_check(tcp_t *);
567 extern void tcp_zcopy_notify(tcp_t *);
568 extern void tcp_get_proto_props(tcp_t *, struct sock_proto_props *);

566 /*
567  * Bind related functions in tcp_bind.c
568  */
569 extern int tcp_bind_check(conn_t *, struct sockaddr *, socklen_t,
570         cred_t *, boolean_t);
571 extern void tcp_bind_hash_insert(tf_t *, tcp_t *, int);
572 extern void tcp_bind_hash_remove(tcp_t *);
573 extern in_port_t tcp_bindi(tcp_t *, in_port_t, const in6_addr_t *,
574         int, boolean_t, boolean_t, boolean_t);
575 extern in_port_t tcp_update_next_port(in_port_t, const tcp_t *,
576         boolean_t);

578 /*
579  * Fusion related functions in tcp_fusion.c.
580  */
581 extern void tcp_fuse(tcp_t *, uchar_t *, tpha_t *);
582 extern void tcp_unfuse(tcp_t *);
583 extern boolean_t tcp_fuse_output(tcp_t *, mblk_t *, uint32_t);
584 extern void tcp_fuse_output_urg(tcp_t *, mblk_t *);
585 extern boolean_t tcp_fuse_rcv_drain(queue_t *, tcp_t *, mblk_t **);
586 extern size_t tcp_fuse_set_rcv_hiwat(tcp_t *, size_t);
587 extern int tcp_fuse_maxpsz(tcp_t *);
588 extern void tcp_fuse_backenable(tcp_t *);
589 extern void tcp_iss_key_init(uint8_t *, int, tcp_stack_t *);

591 /*

```

```

592 * Output related functions in tcp_output.c.
593 */
594 extern void tcp_close_output(void *, mblk_t *, void *, ip_recv_attr_t *);
595 extern void tcp_output(void *, mblk_t *, void *, ip_recv_attr_t *);
596 extern void tcp_output_urgent(void *, mblk_t *, void *, ip_recv_attr_t *);
597 extern void tcp_rexmit_after_error(tcp_t *);
598 extern void tcp_sack_rexmit(tcp_t *, uint_t *);
599 extern void tcp_send_data(tcp_t *, mblk_t *);
600 extern void tcp_send_synack(void *, mblk_t *, void *, ip_recv_attr_t *);
601 extern void tcp_shutdown_output(void *, mblk_t *, void *, ip_recv_attr_t *);
602 extern void tcp_ss_rexmit(tcp_t *);
603 extern void tcp_update_xmit_tail(tcp_t *, uint32_t);
604 extern void tcp_wput(queue_t *, mblk_t *);
605 extern void tcp_wput_data(tcp_t *, mblk_t *, boolean_t);
606 extern void tcp_wput_sock(queue_t *, mblk_t *);
607 extern void tcp_wput_fallback(queue_t *, mblk_t *);
608 extern void tcp_xmit_ctl(char *, tcp_t *, uint32_t, uint32_t, int);
609 extern void tcp_xmit_listeners_reset(mblk_t *, ip_recv_attr_t *,
610 ip_stack_t *i, conn_t *);
611 extern mblk_t *tcp_xmit_mp(tcp_t *, mblk_t *, int32_t, int32_t *,
612 mblk_t **, uint32_t, boolean_t, uint32_t *, boolean_t);

614 /*
615 * Input related functions in tcp_input.c.
616 */
617 extern void tcp_icmp_input(void *, mblk_t *, void *, ip_recv_attr_t *);
618 extern void tcp_input_data(void *, mblk_t *, void *, ip_recv_attr_t *);
619 extern void tcp_input_listener_unbound(void *, mblk_t *, void *,
620 ip_recv_attr_t *);
621 extern boolean_t tcp_paws_check(tcp_t *, tcpha_t *, tcp_opt_t *);
622 extern uint_t tcp_rcv_drain(tcp_t *);
623 extern void tcp_rcv_enqueue(tcp_t *, mblk_t *, uint_t, cred_t *);
624 extern boolean_t tcp_verifyicmp(conn_t *, void *, icmp6_t *, icmp6_t *,
625 ip_recv_attr_t *);

627 /*
628 * Kernel socket related functions in tcp_socket.c.
629 */
630 extern int tcp_fallback(sock_lower_handle_t, queue_t *, boolean_t,
631 so_proto_quiesced_cb_t, sock_quiesce_arg_t *);
632 extern boolean_t tcp_newconn_notify(tcp_t *, ip_recv_attr_t *);

634 /*
635 * Timer related functions in tcp_timers.c.
636 */
637 extern void tcp_ack_timer(void *);
638 extern void tcp_close_linger_timeout(void *);
639 extern void tcp_keeplive_timer(void *);
640 extern void tcp_push_timer(void *);
641 extern void tcp_reass_timer(void *);
642 extern mblk_t *tcp_timermp_alloc(int);
643 extern void tcp_timermp_free(tcp_t *);
644 extern timeout_id_t tcp_timeout(conn_t *, void (*)(void *), hrttime_t);
645 extern clock_t tcp_timeout_cancel(conn_t *, timeout_id_t);
646 extern void tcp_timer(void *arg);
647 extern void tcp_timers_stop(tcp_t *);

649 /*
650 * TCP TPI related functions in tcp_tpi.c.
651 */
652 extern void tcp_addr_req(tcp_t *, mblk_t *);
653 extern void tcp_capability_req(tcp_t *, mblk_t *);
654 extern boolean_t tcp_conn_con(tcp_t *, uchar_t *, mblk_t *,
655 mblk_t **, ip_recv_attr_t *);
656 extern void tcp_err_ack(tcp_t *, mblk_t *, int, int);
657 extern void tcp_err_ack_prim(tcp_t *, mblk_t *, int, int, int);

```

```

658 extern void tcp_info_req(tcp_t *, mblk_t *);
659 extern void tcp_send_conn_ind(void *, mblk_t *, void *);
660 extern void tcp_send_pending(void *, mblk_t *, void *, ip_recv_attr_t *);
661 extern void tcp_tpi_accept(queue_t *, mblk_t *);
662 extern void tcp_tpi_bind(tcp_t *, mblk_t *);
663 extern int tcp_tpi_close(queue_t *, int);
664 extern int tcp_tpi_close_accept(queue_t *);
665 extern void tcp_tpi_connect(tcp_t *, mblk_t *);
666 extern int tcp_tpi_opt_get(queue_t *, t_scalar_t, t_scalar_t, uchar_t *);
667 extern int tcp_tpi_opt_set(queue_t *, uint_t, int, int, uint_t, uchar_t *,
668 uint_t *, uchar_t *, void *, cred_t *);
669 extern void tcp_tpi_unbind(tcp_t *, mblk_t *);
670 extern void tcp_tli_accept(tcp_t *, mblk_t *);
671 extern void tcp_use_pure_tpi(tcp_t *);
672 extern void tcp_do_capability_ack(tcp_t *, struct T_capability_ack *,
673 t_uscalar_t);

675 /*
676 * TCP option processing related functions in tcp_opt_data.c
677 */
682 extern int tcp_opt_default(queue_t *, t_scalar_t, t_scalar_t, uchar_t *);
678 extern int tcp_opt_get(conn_t *, int, int, uchar_t *);
679 extern int tcp_opt_set(conn_t *, uint_t, int, int, uint_t, uchar_t *,
680 uint_t *, uchar_t *, void *, cred_t *);

682 /*
683 * TCP time wait processing related functions in tcp_time_wait.c.
684 */
685 extern void tcp_time_wait_append(tcp_t *);
686 extern void tcp_time_wait_collector(void *);
687 extern boolean_t tcp_time_wait_remove(tcp_t *, tcp_squeue_priv_t *);
688 extern void tcp_time_wait_processing(tcp_t *, mblk_t *, uint32_t,
689 uint32_t, int, tcpha_t *, ip_recv_attr_t *);

691 /*
692 * Misc functions in tcp_misc.c.
693 */
694 extern uint32_t tcp_find_listener_conf(tcp_stack_t *, in_port_t);
695 extern void tcp_ioctl_abort_conn(queue_t *, mblk_t *);
696 extern void tcp_listener_conf_cleanup(tcp_stack_t *);
697 extern void tcp_stack_cpu_add(tcp_stack_t *, processorid_t);

699 #endif /* _KERNEL */

701 #ifdef __cplusplus
702 }

```

unchanged portion omitted

new/usr/src/uts/common/inet/tcp_stats.h

1

```
*****
7689 Wed Aug 8 12:51:12 2012
new/usr/src/uts/common/inet/tcp_stats.h
3065 some functions in the tcp module can be static
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
24 */

26 #ifndef _INET_TCP_STATS_H
27 #define _INET_TCP_STATS_H

29 /*
30  * TCP private kernel statistics declarations.
31 */

33 #ifdef __cplusplus
34 extern "C" {
35 #endif

37 #ifdef _KERNEL

39 /*
40  * TCP Statistics.
41  *
42  * How TCP statistics work.
43  *
44  * There are two types of statistics invoked by two macros.
45  *
46  * TCP_STAT(name) does non-atomic increment of a named stat counter. It is
47  * supposed to be used in non MT-hot paths of the code.
48  *
49  * TCP_DBGSTAT(name) does atomic increment of a named stat counter. It is
50  * supposed to be used for DEBUG purposes and may be used on a hot path.
51  * These counters are only available in a debugged kernel. They are grouped
52  * These counters are only available in a debugged kernel. They are grouped
53  * under the TCP_DEBUG_COUNTER C pre-processor condition.
54  *
55  * Both TCP_STAT and TCP_DBGSTAT counters are available using kstat
56  * (use "kstat tcp" to get them).
57  *
58  * How to add new counters.
59  *
60  * 1) Add a field in the tcp_stat structure describing your counter.
61  * 2) Add a line in the template in tcp_kstat2_init() with the name
```

new/usr/src/uts/common/inet/tcp_stats.h

2

```
61  * of the counter.
62  * 3) Update tcp_clr_stats() and tcp_cp_stats() with the new counters.
63  * IMPORTANT!! - make sure that all the above functions are in sync !!
64  * 4) Use either TCP_STAT or TCP_DBGSTAT with the name.
65  *
66  * Please avoid using private counters which are not kstat-exported.
67  *
68  * Implementation note.
69  *
70  * Both the MIB2 and tcp_stat_t counters are kept per CPU in the array
71  * tcps_sc in tcp_stack_t. Each array element is a pointer to a
72  * tcp_stats_cpu_t struct. Once allocated, the tcp_stats_cpu_t struct is
73  * not freed until the tcp_stack_t is going away. So there is no need to
74  * acquire a lock before accessing the stats counters.
75 */

77 #ifndef TCP_DEBUG_COUNTER
78 #ifdef DEBUG
79 #define TCP_DEBUG_COUNTER 1
80 #else
81 #define TCP_DEBUG_COUNTER 0
82 #endif
83 #endif

85 /* Kstats */
86 typedef struct tcp_stat {
87     kstat_named_t    tcp_time_wait_syn_success;
88     kstat_named_t    tcp_clean_death_nondetached;
89     kstat_named_t    tcp_eager_blowoff_q;
90     kstat_named_t    tcp_eager_blowoff_q0;
91     kstat_named_t    tcp_no_listener;
92     kstat_named_t    tcp_listendrop;
93     kstat_named_t    tcp_listendropq0;
94     kstat_named_t    tcp_wsrv_called;
95     kstat_named_t    tcp_flwctl_on;
96     kstat_named_t    tcp_timer_fire_early;
97     kstat_named_t    tcp_timer_fire_miss;
98     kstat_named_t    tcp_zcopy_on;
99     kstat_named_t    tcp_zcopy_off;
100    kstat_named_t    tcp_zcopy_backoff;
101    kstat_named_t    tcp_fusion_flowctl;
102    kstat_named_t    tcp_fusion_backenabed;
103    kstat_named_t    tcp_fusion_urg;
104    kstat_named_t    tcp_fusion_putnext;
105    kstat_named_t    tcp_fusion_unfusable;
106    kstat_named_t    tcp_fusion_aborted;
107    kstat_named_t    tcp_fusion_unqualified;
108    kstat_named_t    tcp_fusion_rrw_busy;
109    kstat_named_t    tcp_fusion_rrw_msgcnt;
110    kstat_named_t    tcp_fusion_rrw_plugged;
111    kstat_named_t    tcp_in_ack_unsent_drop;
112    kstat_named_t    tcp_sock_fallback;
113    kstat_named_t    tcp_lso_enabled;
114    kstat_named_t    tcp_lso_disabled;
115    kstat_named_t    tcp_lso_times;
116    kstat_named_t    tcp_lso_pkt_out;
117    kstat_named_t    tcp_listen_cnt_drop;
118    kstat_named_t    tcp_listen_mem_drop;
119    kstat_named_t    tcp_zwin_mem_drop;
120    kstat_named_t    tcp_zwin_ack_syn;
121    kstat_named_t    tcp_rst_unsent;
122    kstat_named_t    tcp_reclaim_cnt;
123    kstat_named_t    tcp_reass_timeout;
124 #ifdef TCP_DEBUG_COUNTER
125     kstat_named_t    tcp_time_wait;
126     kstat_named_t    tcp_rput_time_wait;
```

```
127     kstat_named_t    tcp_detach_time_wait;
128     kstat_named_t    tcp_timeout_calls;
129     kstat_named_t    tcp_timeout_cached_alloc;
130     kstat_named_t    tcp_timeout_cancel_reqs;
131     kstat_named_t    tcp_timeout_canceled;
132     kstat_named_t    tcp_timermp_freed;
133     kstat_named_t    tcp_push_timer_cnt;
134     kstat_named_t    tcp_ack_timer_cnt;
135 #endif
136 } tcp_stat_t;
    unchanged_portion_omitted
```