```
**********************************************************
    4265 Tue Oct  2 19:27:28 2012
new/usr/src/cmd/cmd-inet/usr.lib/in.ndpd/defs.h
3245 in.ndp daemon should not be session leader
Reviewed by: Sebastien Roy <sebastien.roy@delphix.com>
Reviewed by: Garrett D'Amore <garrett@damore.org>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _NDPD_DEFS_H
#define _NDPD_DEFS_H

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <errno.h>
#include <sys/types.h>
#include <fcntl.h>
#include <signal.h>
#include <poll.h>
#include <unistd.h>
#include <time.h>
#include <stdarg.h>
#include <syslog.h>

#include <sys/param.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netdb.h>

#include <sys/ioctl.h>
#include <sys/sockio.h>
#include <net/if.h>
#include <sys/stropts.h>

#include <string.h>
#include <ctype.h>

#include <netinet/in_systm.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netinet/if_ether.h>
```



```
#include <netinet/ip6.h>
#include <netinet/icmp6.h>
#include <net/route.h>
#include <libipadm.h>
#include <ipadm_ndpd.h>

#include "tables.h"

#ifdef  __cplusplus
extern "C" {
#endif

#define CURHOP_UNSPECIFIED 0
#define PATH_NDPD_CONF  "/etc/inet/ndpd.conf"
#define PATH_PID        "/var/run/in.ndpd.pid"

extern int debug, no_loopback;

extern struct in6_addr all_nodes_mcast;
extern struct in6_addr all_routers_mcast;

extern int                      rtsock;
extern struct   rt_msghdr       *rt_msg;
extern struct   sockaddr_in6    *rta_gateway;
extern struct   sockaddr_dl     *rta_ifp;

/* Debug flags */
#define D_ALL           0xffff
#define D_DEFAULTS      0x0001          /* Default values in config file */
#define D_CONFIG        0x0002          /* Config file */
#define D_PHYINT        0x0004          /* phyint table */
#define D_PREFIX        0x0008          /* prefix table */
#define D_ROUTER        0x0010          /* router table */
#define D_STATE         0x0020          /* RS/RA state machine */
#define D_IFSCAN        0x0040          /* Scan of kernel interfaces */
#define D_TIMER         0x0080          /* Timer mechanism */
#define D_PARSE         0x0100          /* config file parser */
#define D_PKTIN         0x0200          /* Received packet */
#define D_PKTBAD        0x0400          /* Malformed packet */
#define D_PKTOUT        0x0800          /* Sent packet */
#define D_TMP           0x1000          /* RFC3041 mechanism */
#define D_DHCP          0x2000          /* RFC3315 DHCPv6 (stateful addrs) */

#define IF_SEPARATOR            ':'
#define IPV6_MAX_HOPS           255
#define IPV6_MIN_MTU            (1024+256)
#define IPV6_ABITS              128
#define TMP_TOKEN_BITS          64
#define TMP_TOKEN_BYTES         (TMP_TOKEN_BITS / 8)
#define MAX_DAD_FAILURES        5

/* Return a random number from a an range inclusive of the endpoints */
#define GET_RANDOM(LOW, HIGH) (random() % ((HIGH) - (LOW) + 1) + (LOW))

#define TIMER_INFINITY  0xFFFFFFFFU     /* Never time out */
#define PREFIX_INFINITY 0XFFFFFFFFU     /* A "forever" prefix lifetime */

/*
 * Used by 2 hour rule for stateless addrconf
 */
#define MIN_VALID_LIFETIME      (2*60*60)               /* In seconds */

/*
 * Control how often pi_ReachableTime gets re-randomized
 */
#define MIN_REACH_RANDOM_INTERVAL       (60*1000)       /* 1 minute in ms */
```

```
125 #define MAX_REACH_RANDOM_INTERVAL       (60*60*1000)    /* 1 hour in ms */

127 /*
128  * Parsing constants
129  */
130 #define MAXLINELEN      4096
131 #define MAXARGSPERLINE  128

133 void            timer_schedule(uint_t delay);
134 extern void     logmsg(int level, const char *fmt, ...);
135 extern void     logperror(const char *str);
136 extern void     logperror_pi(const struct phyint *pi, const char *str);
137 extern void     logperror_pr(const struct prefix *pr, const char *str);
138 extern int      parse_config(char *config_file, boolean_t file_required);

140 extern int      poll_add(int fd);
141 extern int      poll_remove(int fd);

143 extern char     *fmt_lla(char *llabuf, int bufsize, uchar_t *lla, int llalen);

145 extern int      do_dad(char *ifname, struct sockaddr_in6 *testaddr);

147 #ifdef  __cplusplus
148 }
```

unchanged_portion_omitted_

```
**********************************************************
   66314 Tue Oct  2 19:27:28 2012
new/usr/src/cmd/cmd-inet/usr.lib/in.ndpd/main.c
3245 in.ndp daemon should not be session leader
Reviewed by: Sebastien Roy <sebastien.roy@delphix.com>
Reviewed by: Garrett D'Amore <garrett@damore.org>
**********************************************************
______unchanged_portion_omitted_

 908 static void
 909 daemonize_ndpd(void)
 910 {
 911         FILE *pidfp;
 912         mode_t pidmode = (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH); /* 0644 */
 911         struct itimerval it;
 912         boolean_t timerval = _B_TRUE;

 914         /*
 915          * Need to get current timer settings so they can be restored
 916          * after the fork(), as the it_value and it_interval values for
 917          * the ITIMER_REAL timer are reset to 0 in the child process.
 918          */
 919         if (getitimer(ITIMER_REAL, &it) < 0) {
 920                 if (debug & D_TIMER)
 921                         logmsg(LOG_DEBUG,
 922                             "daemonize_ndpd: failed to get itimerval\n");
 923                 timerval = _B_FALSE;
 924         }

 926         /* Daemonize. */
 927         if (daemon(0, 0) == -1) {
 929         switch (fork()) {
 930         case 0:
 931                 /* Child */
 932                 break;
 933         case -1:
 928                 logperror("fork");
 929                 exit(1);
 936         default:
 937                 /* Parent */
 938                 _exit(0);
 930         }

 941         /* Store our process id, blow away any existing file if it exists. */
 942         if ((pidfp = fopen(PATH_PID, "w")) == NULL) {
 943                 (void) fprintf(stderr, "%s: unable to open " PATH_PID ": %s\n",
 944                     argv0[0], strerror(errno));
 945         } else {
 946                 (void) fprintf(pidfp, "%ld\n", getpid());
 947                 (void) fclose(pidfp);
 948                 (void) chmod(PATH_PID, pidmode);
 949         }

 951         (void) close(0);
 952         (void) close(1);
 953         (void) close(2);

 955         (void) chdir("/");
 956         (void) open("/dev/null", O_RDWR);
 957         (void) dup2(0, 1);
 958         (void) dup2(0, 2);
 959         (void) setsid();

 932         already_daemonized = _B_TRUE;

 934         /*
```



```
 935          * Restore timer values, if we were able to save them; if not,
 936          * check and set the right value by calling run_timeouts().
 937          */
 938         if (timerval) {
 939                 if (setitimer(ITIMER_REAL, &it, NULL) < 0) {
 940                         logperror("daemonize_ndpd: setitimer");
 941                         exit(2);
 942                 }
 943         } else {
 944                 run_timeouts();
 945         }
 946 }
______unchanged_portion_omitted_

1324 /*
1325  * Pick up a signal "byte" from the pipe and process it.
1326  */
1327 static void
1328 in_signal(int fd)
1329 {
1330         uchar_t buf;
1331         struct phyint *pi;
1332         struct phyint *next_pi;

1334         switch (read(fd, &buf, sizeof (buf))) {
1335         case -1:
1336                 logperror("in_signal: read");
1337                 exit(1);
1338                 /* NOTREACHED */
1339         case 1:
1340                 break;
1341         case 0:
1342                 logmsg(LOG_ERR, "in_signal: read eof\n");
1343                 exit(1);
1344                 /* NOTREACHED */
1345         default:
1346                 logmsg(LOG_ERR, "in_signal: read > 1\n");
1347                 exit(1);
1348         }

1350         if (debug & D_TIMER)
1351                 logmsg(LOG_DEBUG, "in_signal() got %d\n", buf);

1353         switch (buf) {
1354         case SIGALRM:
1355                 if (debug & D_TIMER) {
1356                         uint_t now = getcurrenttime();

1358                         logmsg(LOG_DEBUG, "in_signal(SIGALRM) delta %u\n",
1359                             now - timer_next);
1360                 }
1361                 timer_next = TIMER_INFINITY;
1362                 run_timeouts();
1363                 break;
1364         case SIGHUP:
1365                 /* Re-read config file by exec'ing ourselves */
1366                 for (pi = phyints; pi != NULL; pi = next_pi) {
1367                         next_pi = pi->pi_next;
1368                         if (pi->pi_AdvSendAdvertisements)
1369                                 check_to_advertise(pi, START_FINAL_ADV);

1371                         /*
1372                          * Remove all the configured addresses.
1373                          * Remove the addrobj names created with ipmgmtd.
1374                          * Release the dhcpv6 addresses if any.
1375                          * Cleanup the phyints.
```

```
1376                            */
1377                           phyint_delete(pi);
1378                   }

1380                   /*
1381                    * Prevent fd leaks.  Everything gets re-opened at start-up
1382                    * time.  0, 1, and 2 are closed and re-opened as
1383                    * /dev/null, so we'll leave those open.
1384                    */
1385                   closefrom(3);

1387                   logmsg(LOG_ERR, "SIGHUP: restart and reread config file\n");
1388                   (void) execv(argv0[0], argv0);
1418                   (void) unlink(PATH_PID);
1389                   _exit(0177);
1390                   /* NOTREACHED */
1391           case SIGUSR1:
1392                   logmsg(LOG_DEBUG, "Printing configuration:\n");
1393                   phyint_print_all();
1394                   break;
1395           case SIGINT:
1396           case SIGTERM:
1397           case SIGQUIT:
1398                   for (pi = phyints; pi != NULL; pi = next_pi) {
1399                           next_pi = pi->pi_next;
1400                           if (pi->pi_AdvSendAdvertisements)
1401                                   check_to_advertise(pi, START_FINAL_ADV);

1403                           phyint_delete(pi);
1404                   }
1405                   (void) unlink(NDPD_SNMP_SOCKET);
1436                   (void) unlink(PATH_PID);
1406                   exit(0);
1407                   /* NOTREACHED */
1408           case 255:
1409                   /*
1410                    * Special "signal" from loopback_ra_enqueue.
1411                    * Handle any queued loopback router advertisements.
1412                    */
1413                   loopback_ra_dequeue();
1414                   break;
1415           default:
1416                   logmsg(LOG_ERR, "in_signal: unknown signal: %d\n", buf);
1417           }
1418 }
```

_____***unchanged_portion_omitted_***