```
**********************************************************
   13901 Wed Jan 30 11:20:30 2019
new/usr/src/cmd/ttymon/tmhandler.c
10143 smatch fix for ttymon
**********************************************************
```

```
   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, Version 1.0 only
   6  * (the "License").  You may not use this file except in compliance
   7  * with the License.
   8  *
   9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
  10  * or http://www.opensolaris.org/os/licensing.
  11  * See the License for the specific language governing permissions
  12  * and limitations under the License.
  13  *
  14  * When distributing Covered Code, include this CDDL HEADER in each
  15  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  16  * If applicable, add the following below this CDDL HEADER, with the
  17  * fields enclosed by brackets "[]" replaced with your own identifying
  18  * information: Portions Copyright [yyyy] [name of copyright owner]
  19  *
  20  * CDDL HEADER END
  21  */
  22 /*
  23  * Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
  24  * Use is subject to license terms.
  25  */
  26 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
  27 /*        All Rights Reserved   */
  28 /*
  29  * Copyright (c) 2018, Joyent, Inc.
  30  */


  30 #pragma ident   "%Z%%M% %I%     %E% SMI"


  33 #include <stdlib.h>
  34 #include <stdio.h>
  35 #include <fcntl.h>
  36 #include <errno.h>
  37 #include <poll.h>
  38 #include <string.h>
  39 #include <termio.h>
  40 #include <signal.h>
  41 #include <sys/types.h>
  42 #include <sys/stropts.h>
  43 #include <unistd.h>
  44 #include <sys/wait.h>
  45 #include "ttymon.h"
  46 #include "tmstruct.h"
  47 #include "tmextern.h"
  48 #include "sac.h"

  50 extern  int     Retry;
  51 static  struct  pmtab   *find_pid();
  52 static  void    kill_children();

  54 static  struct  pmtab   *find_fd();
  55 static  void    pcsync_close();
  56 extern  void    sigalarm();
  57 extern  void    tmchild();
```



```
  59 /*
  60  *      fork_tmchild    - fork child on the device
  61  */
  62 static  void
  63 fork_tmchild(pmptr)
  64 struct  pmtab   *pmptr;
  65 {
  66         pid_t   pid;
  67         sigset_t        cset;
  68         sigset_t        tset;
  69         int     pcpipe0[2], pcpipe1[2];
  70         int     p0;

  72 #ifdef  DEBUG
  73         debug("in fork_tmchild");
  74 #endif
  75         pmptr->p_inservice = FALSE;

  77         /*
  78          * initialize pipe.
  79          * Child has pcpipe[0] pipe fd for reading and writing
  80          * and closes pcpipe[1]. Parent has pcpipe[1] pipe fd for
  81          * reading and writing and closes pcpipe[0].
  82          *
  83          * This way if the child process exits the parent's block
  84          * read on pipe will return immediately as the other end of
  85          * the pipe has closed. Similarly if the parent process exits
  86          * child's blocking read on the pipe will return immediately.
  87          */

  89         if (((p0 = pipe(pcpipe0)) == -1) || (pipe(pcpipe1) == -1))  {
  90                 if (p0 == 0) {
  91                         close(pcpipe0[0]);
  92                         close(pcpipe0[1]);
  93                 }
  94                 log("pipe() failed: %s", strerror(errno));
  95                 pmptr->p_status = VALID;
  96                 pmptr->p_pid = 0;
  97                 Retry = TRUE;
  98         }

 100         /* protect following region from SIGCLD */
 101         (void)sigprocmask(SIG_SETMASK, NULL, &cset);
 102         tset = cset;
 103         (void)sigaddset(&tset, SIGCLD);
 104         (void)sigprocmask(SIG_SETMASK, &tset, NULL);
 105         if( (pid=fork()) == 0 ) {
 106                 /*
 107                  * Close all file descriptors except pmptr->p_fd
 108                  * Wait for the parent process to close its fd
 109                  */
 110                 pcsync_close(pcpipe0, pcpipe1, pid, pmptr->p_fd);
 111                 /* The CHILD */
 112                 tmchild(pmptr);
 113                 /* tmchild should never return */
 114                 fatal("tmchild for <%s> returns unexpected", pmptr->p_device);
 115         }
 116         else if (pid < 0) {
 117                 log("fork failed: %s", strerror(errno));
 118                 pmptr->p_status = VALID;
 119                 pmptr->p_pid = 0;
 120                 Retry = TRUE;
 121         }
 122         else {
 123                 /*
 124                  * The PARENT - store pid of child and close the device
```

```
125                  */
126                 pmptr->p_pid = pid;
127         }
128         if (pmptr->p_fd > 0) {
129                 (void)close(pmptr->p_fd);
130                 pmptr->p_fd = 0;
131         }
132         (void)sigprocmask(SIG_SETMASK, &cset, NULL);
133         /*
134          * Wait for child to close file descriptors
135          */
136         pcsync_close(pcpipe0, pcpipe1, pid, pmptr->p_fd);
137 }
______unchanged_portion_omitted_

628 /*
629  * pcsync_close -  For the child process close all open fd's except
630  * the one that is passed to the routine. Coordinate the reads and
631  * writes to the pipes by the parent and child process to ensure
632  * the parent and child processes have closed all the file descriptors
633  * that are not needed any more.
634  */
635 static void
636 pcsync_close(int *p0, int *p1, int pid, int fd)
636 pcsync_close(p0, p1, pid, fd)
637 int     *p0;
638 int     *p1;
639 int     pid;
637 {
638         char    ch;

640         if (pid == 0) {                         /* Child */
641                 struct  pmtab   *tp;
642                 for (tp = PMtab; tp; tp = tp->p_next)
643                         if ((tp->p_fd > 0) && (tp->p_fd != fd))
644                                 close(tp->p_fd);
645                 close(p0[1]); close(p1[0]);
646                 if (read(p0[0], &ch, 1) == 1)
647                         write(p1[1], "a", 1);
648                 close(p0[0]); close(p1[1]);
649         } else {                                /* Parent */
650                 close(p0[0]); close(p1[1]);
651                 if (write(p0[1], "a", 1) == 1)
652                         read(p1[0], &ch, 1);
653                 close(p0[1]); close(p1[0]);
654         }
655 }
______unchanged_portion_omitted_
```