```
**********************************************************
   11080 Thu Jul 27 16:35:41 2017
new/usr/src/head/limits.h
8527 tty buffer/queue sizes should be larger
**********************************************************
```

```
/*
 * 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 (c) 2013 Gary Mills
 * Copyright 2017 RackTop Systems.
 *
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*      Copyright (c) 1988 AT&T */
/*        All Rights Reserved   */


#ifndef _LIMITS_H
#define _LIMITS_H

#include <sys/feature_tests.h>
#include <sys/isa_defs.h>
#include <iso/limits_iso.h>

/*
 * Include fixed width type limits as proposed by the ISO/JTC1/SC22/WG14 C
 * committee's working draft for the revision of the current ISO C standard,
 * ISO/IEC 9899:1990 Programming language - C.  These are not currently
 * required by any standard but constitute a useful, general purpose set
 * of type definitions and limits which is namespace clean with respect to
 * all standards.
 */
#if defined(__EXTENSIONS__) || !defined(_STRICT_STDC) || \
        defined(__XOPEN_OR_POSIX)
#include <sys/int_limits.h>
#endif

#ifdef  __cplusplus
extern "C" {
#endif

#if defined(__EXTENSIONS__) || !defined(_STRICT_STDC) || \
        defined(__XOPEN_OR_POSIX)

#define SSIZE_MAX       LONG_MAX        /* max value of an "ssize_t" */
```



```
/*
 * ARG_MAX is calculated as follows:
 * NCARGS - space for other stuff on initial stack
 * like aux vectors, saved registers, etc..
 */
#define _ARG_MAX32      1048320 /* max length of args to exec 32-bit program */
#define _ARG_MAX64      2096640 /* max length of args to exec 64-bit program */
#ifdef  _LP64
#define ARG_MAX         _ARG_MAX64      /* max length of arguments to exec */
#else   /* _LP64 */
#define ARG_MAX         _ARG_MAX32      /* max length of arguments to exec */
#endif  /* _LP64 */

/*
 * These two symbols have their historical values, the actual buffer is
 * larger.
 */
#endif /* ! codereview */
#ifndef MAX_CANON
#define MAX_CANON       256     /* max bytes in line for canonical processing */
#endif

#ifndef MAX_INPUT
#define MAX_INPUT       512     /* max size of a char input buffer */
#endif

#define NGROUPS_MAX     16      /* max number of groups for a user */

#ifndef PATH_MAX
#define PATH_MAX        1024    /* max # of characters in a path name */
#endif

#define SYMLINK_MAX     1024    /* max # of characters a symlink can contain */

#define PIPE_BUF        5120    /* max # bytes atomic in write to a pipe */

#ifndef TMP_MAX
#define TMP_MAX         17576   /* 26 * 26 * 26 */
#endif

/*
 * POSIX conformant definitions - An implementation may define
 * other symbols which reflect the actual implementation. Alternate
 * definitions may not be as restrictive as the POSIX definitions.
 */
#define _POSIX_AIO_LISTIO_MAX       2
#define _POSIX_AIO_MAX              1
#define _POSIX_ARG_MAX           4096
#ifdef _XPG6
#define _POSIX_CHILD_MAX           25
#else
#define _POSIX_CHILD_MAX            6   /* POSIX.1-1990 default */
#endif
#define _POSIX_CLOCKRES_MIN     20000000
#define _POSIX_DELAYTIMER_MAX      32
#define _POSIX_LINK_MAX             8
#define _POSIX_MAX_CANON          255
#define _POSIX_MAX_INPUT          255
#define _POSIX_MQ_OPEN_MAX          8
#define _POSIX_MQ_PRIO_MAX         32
#define _POSIX_NAME_MAX            14
#ifdef _XPG6
#define _POSIX_NGROUPS_MAX          8
#define _POSIX_OPEN_MAX            20
```

```
#define _POSIX_PATH_MAX           256
#else                                   /* POSIX.1-1990 defaults */
#define _POSIX_NGROUPS_MAX          0
#define _POSIX_OPEN_MAX            16
#define _POSIX_PATH_MAX           255
#endif
#define _POSIX_PIPE_BUF           512
#define _POSIX_RTSIG_MAX            8
#define _POSIX_SEM_NSEMS_MAX      256
#define _POSIX_SEM_VALUE_MAX    32767
#define _POSIX_SIGQUEUE_MAX        32
#define _POSIX_SSIZE_MAX        32767
#define _POSIX_STREAM_MAX           8
#define _POSIX_TIMER_MAX           32
#ifdef _XPG6
#define _POSIX_TZNAME_MAX           6
#else
#define _POSIX_TZNAME_MAX           3   /* POSIX.1-1990 default */
#endif
/* POSIX.1c conformant */
#define _POSIX_LOGIN_NAME_MAX                   9
#define _POSIX_THREAD_DESTRUCTOR_ITERATIONS     4
#define _POSIX_THREAD_KEYS_MAX                  128
#define _POSIX_THREAD_THREADS_MAX               64
#define _POSIX_TTY_NAME_MAX                     9
/* UNIX 03 conformant */
#define _POSIX_HOST_NAME_MAX                    255
#define _POSIX_RE_DUP_MAX                       255
#define _POSIX_SYMLINK_MAX                      255
#define _POSIX_SYMLOOP_MAX                      8

/*
 * POSIX.2 and XPG4-XSH4 conformant definitions
 */

#define _POSIX2_BC_BASE_MAX              99
#define _POSIX2_BC_DIM_MAX             2048
#define _POSIX2_BC_SCALE_MAX             99
#define _POSIX2_BC_STRING_MAX          1000
#define _POSIX2_COLL_WEIGHTS_MAX          2
#define _POSIX2_EXPR_NEST_MAX            32
#define _POSIX2_LINE_MAX               2048
#define _POSIX2_RE_DUP_MAX              255
/* UNIX 03 conformant */
#define _POSIX2_CHARCLASS_NAME_MAX       14

#define BC_BASE_MAX             _POSIX2_BC_BASE_MAX
#define BC_DIM_MAX              _POSIX2_BC_DIM_MAX
#define BC_SCALE_MAX            _POSIX2_BC_SCALE_MAX
#define BC_STRING_MAX           _POSIX2_BC_STRING_MAX
#define COLL_WEIGHTS_MAX        10
#define EXPR_NEST_MAX           _POSIX2_EXPR_NEST_MAX
#define LINE_MAX                _POSIX2_LINE_MAX
#if !defined(_XPG6)
#define RE_DUP_MAX              _POSIX2_RE_DUP_MAX
#else
#define RE_DUP_MAX              _POSIX_RE_DUP_MAX
#endif /* !defined(_XPG6) */

#endif /* defined(__EXTENSIONS__) || !defined(_STRICT_STDC) ... */

#if defined(__EXTENSIONS__) || \
        (!defined(_STRICT_STDC) && !defined(_POSIX_C_SOURCE)) || \
        defined(_XOPEN_SOURCE)

/*
```



```
 * For dual definitions for PASS_MAX and sysconf.c
 */
#define _PASS_MAX_XPG   8       /* old standards PASS_MAX */
#define _PASS_MAX       256     /* modern Solaris PASS_MAX */

#if defined(_XPG3) && !defined(_XPG6)
#define PASS_MAX        _PASS_MAX_XPG   /* max # of characters in a password */
#else   /* XPG6 or just Solaris */
#define PASS_MAX        _PASS_MAX       /* max # of characters in a password */
#endif  /* defined(_XPG3) && !defined(_XPG6) */

#define CHARCLASS_NAME_MAX      _POSIX2_CHARCLASS_NAME_MAX

#define NL_ARGMAX       9       /* max value of "digit" in calls to the */
                                /* NLS printf() and scanf() */
#define NL_LANGMAX      14      /* max # of bytes in a LANG name */
#define NL_MSGMAX       32767   /* max message number */
#define NL_NMAX         1       /* max # bytes in N-to-1 mapping characters */
#define NL_SETMAX       255     /* max set number */
#define NL_TEXTMAX      2048    /* max set number */
#define NZERO           20      /* default process priority */

#define WORD_BIT        32      /* # of bits in a "word" or "int" */
#if defined(_LP64)
#define LONG_BIT        64      /* # of bits in a "long" */
#else   /* _ILP32 */
#define LONG_BIT        32      /* # of bits in a "long" */
#endif

/* Marked as LEGACY in SUSv2 and removed in UNIX 03 */
#ifndef _XPG6
#define DBL_DIG         15      /* digits of precision of a "double" */
#define DBL_MAX         1.79769313486231570814527E+308   /* max decimal value */
                                                         /* of a double */
#define FLT_DIG         6               /* digits of precision of a "float" */
#define FLT_MAX         3.40282346638528859811704E+38F   /* max decimal value */
                                                         /* of a "float" */
#endif

/* Marked as LEGACY in SUSv1 and removed in SUSv2 */
#ifndef _XPG5
#define DBL_MIN         2.22507385850720138309023E-308   /* min decimal value */
                                                         /* of a double */
#define FLT_MIN         1.17549435082228750796882E-38F   /* min decimal value */
                                                         /* of a float */
#endif

#endif  /* defined(__EXTENSIONS__) || (!defined(_STRICT_STDC) ... */

#define _XOPEN_IOV_MAX  16      /* max # iovec/process with readv()/writev() */
#define _XOPEN_NAME_MAX 255     /* max # bytes in filename excluding null */
#define _XOPEN_PATH_MAX 1024    /* max # bytes in a pathname */

#define IOV_MAX         _XOPEN_IOV_MAX

#if defined(__EXTENSIONS__) || \
        (!defined(_STRICT_STDC) && !defined(__XOPEN_OR_POSIX))

#define FCHR_MAX        1048576         /* max size of a file in bytes */
#define PID_MAX         999999          /* max value for a process ID */

/*
 * POSIX 1003.1a, section 2.9.5, table 2-5 contains [NAME_MAX] and the
 * related text states:
 *
 * A definition of one of the values from Table 2-5 shall be omitted from the
```

```
 * <limits.h> on specific implementations where the corresponding value is
 * equal to or greater than the stated minimum, but where the value can vary
 * depending on the file to which it is applied. The actual value supported for
 * a specific pathname shall be provided by the pathconf() (5.7.1) function.
 *
 * This is clear that any machine supporting multiple file system types
 * and/or a network should not include this define, regardless of protection
 * by the _POSIX_SOURCE and _POSIX_C_SOURCE flags. We chose to ignore that
 * and provide it anyway for compatibility with other platforms that don't
 * follow the spec as precisely as they should. Its usage is discouraged.
 */
#define NAME_MAX        255

#define CHILD_MAX       25      /* max # of processes per user id */
#ifndef OPEN_MAX
#define OPEN_MAX        256     /* max # of files a process can have open */
#endif

#define PIPE_MAX        5120    /* max # bytes written to a pipe in a write */

#define STD_BLK         1024    /* # bytes in a physical I/O block */
#define UID_MAX         2147483647      /* max value for a user or group ID */
#define USI_MAX         4294967295      /* max decimal value of an "unsigned" */
#define SYSPID_MAX      1       /* max pid of system processes */

#ifndef SYS_NMLN                /* also defined in sys/utsname.h */
#define SYS_NMLN        257     /* 4.0 size of utsname elements */
#endif

#ifndef CLK_TCK

#if !defined(_CLOCK_T) || __cplusplus >= 199711L
#define _CLOCK_T
typedef long    clock_t;
#endif  /* !_CLOCK_T */

extern long _sysconf(int);      /* System Private interface to sysconf() */
#define CLK_TCK ((clock_t)_sysconf(3))  /* 3 is _SC_CLK_TCK */

#endif /* CLK_TCK */

#ifdef  __USE_LEGACY_LOGNAME__
#define LOGNAME_MAX     8       /* max # of characters in a login name */
#else   /* __USE_LEGACY_LOGNAME__ */
#define LOGNAME_MAX     32      /* max # of characters in a login name */
                                /* Increased for illumos */
#endif  /* __USE_LEGACY_LOGNAME__ */
#define LOGIN_NAME_MAX  (LOGNAME_MAX + 1)       /* max buffer size */
#define LOGNAME_MAX_TRAD        8               /* traditional length */
#define LOGIN_NAME_MAX_TRAD     (LOGNAME_MAX_TRAD + 1)  /* and size */

#define TTYNAME_MAX     128     /* max # of characters in a tty name */

#endif  /* if defined(__EXTENSIONS__) || (!defined(_STRICT_STDC) ... */

#if     defined(__EXTENSIONS__) || (_POSIX_C_SOURCE >= 199506L)
#include <sys/unistd.h>

#if !defined(_SIZE_T) || __cplusplus >= 199711L
#define _SIZE_T
#if defined(_LP64) || defined(_I32LPx)
typedef unsigned long size_t;   /* size of something in bytes */
#else
typedef unsigned int  size_t;   /* (historical version) */
#endif
#endif  /* _SIZE_T */
```



```
extern long _sysconf(int);      /* System Private interface to sysconf() */

#define PTHREAD_STACK_MIN       ((size_t)_sysconf(_SC_THREAD_STACK_MIN))
/* Added for UNIX98 conformance */
#define PTHREAD_DESTRUCTOR_ITERATIONS   _POSIX_THREAD_DESTRUCTOR_ITERATIONS
#define PTHREAD_KEYS_MAX                _POSIX_THREAD_KEYS_MAX
#define PTHREAD_THREADS_MAX             _POSIX_THREAD_THREADS_MAX
#endif  /* defined(__EXTENSIONS__) || (_POSIX_C_SOURCE >= 199506L) */

#ifdef  __cplusplus
}
#endif

#endif  /* _LIMITS_H */
```

```
**********************************************************
  143771 Thu Jul 27 16:35:42 2017
new/usr/src/uts/common/io/ldterm.c
8527 tty buffer/queue sizes should be larger
**********************************************************
______unchanged_portion_omitted_

 521 /*
 522  * The following tables are from either u8_textprep.c or uconv.c at
 523  * usr/src/common/unicode/. The tables are used to figure out corresponding
 524  * UTF-8 character byte lengths and also the validity of given character bytes.
 525  */
 526 extern const int8_t u8_number_of_bytes[];
 527 extern const uchar_t u8_masks_tbl[];
 528 extern const uint8_t u8_valid_min_2nd_byte[];
 529 extern const uint8_t u8_valid_max_2nd_byte[];

 531 /*
 532  * Unicode character width definition tables from uwidth.c:
 533  */
 534 extern const ldterm_unicode_data_cell_t ldterm_ucode[][16384];

 536 #ifdef LDDEBUG
 537 int     ldterm_debug = 0;
 538 #define DEBUG1(a)       if (ldterm_debug == 1) printf a
 539 #define DEBUG2(a)       if (ldterm_debug >= 2) printf a /* allocations */
 540 #define DEBUG3(a)       if (ldterm_debug >= 3) printf a /* M_CTL Stuff */
 541 #define DEBUG4(a)       if (ldterm_debug >= 4) printf a /* M_READ Stuff */
 542 #define DEBUG5(a)       if (ldterm_debug >= 5) printf a
 543 #define DEBUG6(a)       if (ldterm_debug >= 6) printf a
 544 #define DEBUG7(a)       if (ldterm_debug >= 7) printf a
 545 #else
 546 #define DEBUG1(a)
 547 #define DEBUG2(a)
 548 #define DEBUG3(a)
 549 #define DEBUG4(a)
 550 #define DEBUG5(a)
 551 #define DEBUG6(a)
 552 #define DEBUG7(a)
 553 #endif          /* LDDEBUG */


 556 /*
 557  * Since most of the buffering occurs either at the stream head or in
 558  * the "message currently being assembled" buffer, we have a
 559  * relatively small input queue, so that blockages above us get
 560  * reflected fairly quickly to the module below us.  We also have a
 561  * small maximum packet size, since you can put a message of that
 562  * size on an empty queue no matter how much bigger than the high
 563  * water mark it is.
 564  */
 565 static struct module_info ldtermmiinfo = {
 566         0x0bad,
 567         "ldterm",
 568         0,
 569         _TTY_BUFSIZ,
 570         _TTY_BUFSIZ,
 569         256,
 570         HIWAT,
 571         LOWAT
 572 };
______unchanged_portion_omitted_

 662 /*
 663  * Line discipline open.
 664  */
```



```
 665 /* ARGSUSED1 */
 666 static int
 667 ldtermopen(queue_t *q, dev_t *devp, int oflag, int sflag, cred_t *crp)
 668 {
 669         ldtermstd_state_t *tp;
 670         mblk_t *bp, *qryp;
 671         int len;
 672         struct stroptions *strop;
 673         struct termios *termiosp;
 674         queue_t *wq;

 676         if (q->q_ptr != NULL) {
 677                 return (0);     /* already attached */
 678         }

 680         tp = (ldtermstd_state_t *)kmem_zalloc(sizeof (ldtermstd_state_t),
 681             KM_SLEEP);

 683         /*
 684          * Get termios defaults.  These are stored as
 685          * a property in the "options" node.
 686          */
 687         if (ddi_getlongprop(DDI_DEV_T_ANY, ddi_root_node(), DDI_PROP_NOTPROM,
 688             "ttymodes", (caddr_t)&termiosp, &len) == DDI_PROP_SUCCESS &&
 689             len == sizeof (struct termios)) {
 690                 tp->t_modes = *termiosp;
 691                 tp->t_amodes = *termiosp;
 692                 kmem_free(termiosp, len);
 693         } else {
 694                 /*
 695                  * Gack!  Whine about it.
 696                  */
 697                 cmn_err(CE_WARN, "ldterm: Couldn't get ttymodes property!");
 698         }
 699         bzero(&tp->t_dmodes, sizeof (struct termios));

 701         tp->t_state = 0;

 703         tp->t_line = 0;
 704         tp->t_col = 0;

 706         tp->t_rocount = 0;
 707         tp->t_rocol = 0;

 709         tp->t_message = NULL;
 710         tp->t_endmsg = NULL;
 711         tp->t_msglen = 0;
 712         tp->t_rd_request = 0;

 714         tp->t_echomp = NULL;
 715         tp->t_iocid = 0;
 716         tp->t_wbufcid = 0;
 717         tp->t_vtid = 0;

 719         q->q_ptr = (caddr_t)tp;
 720         WR(q)->q_ptr = (caddr_t)tp;
 721         /*
 722          * The following for EUC and also non-EUC codesets:
 723          */
 724         tp->t_codeset = tp->t_eucleft = tp->t_eucign = tp->t_scratch_len = 0;
 725         bzero(&tp->eucwioc, EUCSIZE);
 726         tp->eucwioc.eucw[0] = 1;        /* ASCII mem & screen width */
 727         tp->eucwioc.scrw[0] = 1;
 728         tp->t_maxeuc = 1;       /* the max len in bytes of an EUC char */
 729         tp->t_eucp = NULL;
 730         tp->t_eucp_mp = NULL;
```

```
731         tp->t_eucwarn = 0;      /* no bad chars seen yet */

733         tp->t_csdata = default_cs_data;
734         tp->t_csmethods = cs_methods[LDTERM_CS_TYPE_EUC];

736         qprocson(q);

738         /*
739          * Find out if the module below us does canonicalization; if
740          * so, we won't do it ourselves.
741          */

743         if ((qryp = open_ioctl(q, MC_CANONQUERY)) == NULL)
744                 goto open_abort;

746         /*
747          * Reformulate as an M_CTL message. The actual data will
748          * be in the b_cont field.
749          */
750         qryp->b_datap->db_type = M_CTL;
751         wq = OTHERQ(q);
752         putnext(wq, qryp);

754         /* allocate a TCSBRK ioctl in case we'll need it on close */
755         if ((qryp = open_ioctl(q, TCSBRK)) == NULL)
756                 goto open_abort;
757         tp->t_drainmsg = qryp;
758         if ((bp = open_mblk(q, sizeof (int))) == NULL)
759                 goto open_abort;
760         qryp->b_cont = bp;

762         /*
763          * Find out if the underlying driver supports proper POSIX close
764          * semantics.  If not, we'll have to approximate it using TCSBRK.  If
765          * it does, it will respond with MC_HAS_POSIX, and we'll catch that in
766          * the ldtermrput routine.
767          *
768          * When the ldterm_drain_limit tunable is set to zero, we behave the
769          * same as old ldterm: don't send this new message, and always use
770          * TCSBRK during close.
771          */
772         if (ldterm_drain_limit != 0) {
773                 if ((qryp = open_ioctl(q, MC_POSIXQUERY)) == NULL)
774                         goto open_abort;
775                 qryp->b_datap->db_type = M_CTL;
776                 putnext(wq, qryp);
777         }

779         /* prepare to clear the water marks on close */
780         if ((bp = open_mblk(q, sizeof (struct stroptions))) == NULL)
781                 goto open_abort;
782         tp->t_closeopts = bp;

784         /*
785          * Set the high-water and low-water marks on the stream head
786          * to values appropriate for a terminal.  Also set the "vmin"
787          * and "vtime" values to 1 and 0, turn on message-nondiscard
788          * mode (as we're in ICANON mode), and turn on "old-style
789          * NODELAY" mode.
790          */
791         if ((bp = open_mblk(q, sizeof (struct stroptions))) == NULL)
792                 goto open_abort;
793         strop = (struct stroptions *)bp->b_wptr;
794         strop->so_flags = SO_READOPT|SO_HIWAT|SO_LOWAT|SO_NDELON|SO_ISTTY;
795         strop->so_readopt = RMSGN;
796         strop->so_hiwat = _TTY_BUFSIZ;
```



```
796         strop->so_hiwat = HIWAT;
797         strop->so_lowat = LOWAT;
798         bp->b_wptr += sizeof (struct stroptions);
799         bp->b_datap->db_type = M_SETOPTS;
800         putnext(q, bp);

802         return (0);             /* this can become a controlling TTY */

804 open_abort:
805         qprocsoff(q);
806         q->q_ptr = NULL;
807         WR(q)->q_ptr = NULL;
808         freemsg(tp->t_closeopts);
809         freemsg(tp->t_drainmsg);
810         /* Dump the state structure */
811         kmem_free(tp, sizeof (ldtermstd_state_t));
812         return (EINTR);
813 }
______unchanged_portion_omitted_


1724 /*
1725  * Do canonical mode input; check whether this character is to be
1726  * treated as a special character - if so, check whether it's equal
1727  * to any of the special characters and handle it accordingly.
1728  * Otherwise, just add it to the current line.
1729  */
1730 static mblk_t *
1731 ldterm_docanon(uchar_t c, mblk_t *bpt, size_t ebsize, queue_t *q,
1732     ldtermstd_state_t *tp, int *dofreep)
1733 {
1734         queue_t *wrq = WR(q);
1735         int i;

1737         /*
1738          * If the previous character was the "literal next"
1739          * character, treat this character as regular input.
1740          */
1741         if (tp->t_state & TS_SLNCH)
1742                 goto escaped;

1744         /*
1745          * Setting a special character to NUL disables it, so if this
1746          * character is NUL, it should not be compared with any of
1747          * the special characters.
1748          */
1749         if (c == _POSIX_VDISABLE) {
1750                 tp->t_state &= ~TS_QUOT;
1751                 goto escaped;
1752         }
1753         /*
1754          * If this character is the literal next character, echo it
1755          * as '^', backspace over it, and record that fact.
1756          */
1757         if ((tp->t_modes.c_lflag & IEXTEN) && c == tp->t_modes.c_cc[VLNEXT]) {
1758                 if (tp->t_modes.c_lflag & ECHO)
1759                         ldterm_outstring((unsigned char *)"^\b", 2, wrq,
1760                             ebsize, tp);
1761                 tp->t_state |= TS_SLNCH;
1762                 goto out;
1763         }
1764         /*
1765          * Check for the editing character. If the display width of
1766          * the last byte at the canonical buffer is not one and also
1767          * smaller than or equal to UNKNOWN_WIDTH, the character at
1768          * the end of the buffer is a multi-byte and/or multi-column
```

```
1769          * character.
1770          */
1771         if (c == tp->t_modes.c_cc[VERASE] || c == tp->t_modes.c_cc[VERASE2]) {
1772                 if (tp->t_state & TS_QUOT) {
1773                         /*
1774                          * Get rid of the backslash, and put the
1775                          * erase character in its place.
1776                          */
1777                         ldterm_erase(wrq, ebsize, tp);
1778                         bpt = tp->t_endmsg;
1779                         goto escaped;
1780                 } else {
1781                         if ((tp->t_state & TS_MEUC) && tp->t_msglen &&
1782                             (*(tp->t_eucp - 1) != 1 &&
1783                             *(tp->t_eucp - 1) <= UNKNOWN_WIDTH))
1784                                 ldterm_csi_erase(wrq, ebsize, tp);
1785                         else
1786                                 ldterm_erase(wrq, ebsize, tp);
1787                         bpt = tp->t_endmsg;
1788                         goto out;
1789                 }
1790         }
1791         if ((tp->t_modes.c_lflag & IEXTEN) && c == tp->t_modes.c_cc[VWERASE]) {
1792                 /*
1793                  * Do "ASCII word" or "multibyte character token/chunk" erase.
1794                  */
1795                 if (tp->t_state & TS_MEUC)
1796                         ldterm_csi_werase(wrq, ebsize, tp);
1797                 else
1798                         ldterm_werase(wrq, ebsize, tp);
1799                 bpt = tp->t_endmsg;
1800                 goto out;
1801         }
1802         if (c == tp->t_modes.c_cc[VKILL]) {
1803                 if (tp->t_state & TS_QUOT) {
1804                         /*
1805                          * Get rid of the backslash, and put the kill
1806                          * character in its place.
1807                          */
1808                         ldterm_erase(wrq, ebsize, tp);
1809                         bpt = tp->t_endmsg;
1810                         goto escaped;
1811                 } else {
1812                         ldterm_kill(wrq, ebsize, tp);
1813                         bpt = tp->t_endmsg;
1814                         goto out;
1815                 }
1816         }
1817         if ((tp->t_modes.c_lflag & IEXTEN) && c == tp->t_modes.c_cc[VREPRINT]) {
1818                 ldterm_reprint(wrq, ebsize, tp);
1819                 goto out;
1820         }
1821         /*
1822          * If the preceding character was a backslash: if the current
1823          * character is an EOF, get rid of the backslash and treat
1824          * the EOF as data; if we're in XCASE mode and the current
1825          * character is part of a backslash-X escape sequence,
1826          * process it; otherwise, just treat the current character
1827          * normally.
1828          */
1829         if (tp->t_state & TS_QUOT) {
1830                 tp->t_state &= ~TS_QUOT;
1831                 if (c == tp->t_modes.c_cc[VEOF]) {
1832                         /*
1833                          * EOF character. Since it's escaped, get rid
1834                          * of the backslash and put the EOF character
```



```
1835                          * in its place.
1836                          */
1837                         ldterm_erase(wrq, ebsize, tp);
1838                         bpt = tp->t_endmsg;
1839                 } else {
1840                         /*
1841                          * If we're in XCASE mode, and the current
1842                          * character is part of a backslash-X
1843                          * sequence, get rid of the backslash and
1844                          * replace the current character with what
1845                          * that sequence maps to.
1846                          */
1847                         if ((tp->t_modes.c_lflag & XCASE) &&
1848                             imaptab[c] != '\0') {
1849                                 ldterm_erase(wrq, ebsize, tp);
1850                                 bpt = tp->t_endmsg;
1851                                 c = imaptab[c];
1852                         }
1853                 }
1854         } else {
1855                 /*
1856                  * Previous character wasn't backslash; check whether
1857                  * this was the EOF character.
1858                  */
1859                 if (c == tp->t_modes.c_cc[VEOF]) {
1860                         /*
1861                          * EOF character. Don't echo it unless
1862                          * ECHOCTL is set, don't stuff it in the
1863                          * current line, but send the line up the
1864                          * stream.
1865                          */
1866                         if ((tp->t_modes.c_lflag & ECHOCTL) &&
1867                             (tp->t_modes.c_lflag & IEXTEN) &&
1868                             (tp->t_modes.c_lflag & ECHO)) {
1869                                 i = ldterm_echo(c, wrq, ebsize, tp);
1870                                 while (i > 0) {
1871                                         ldterm_outchar('\b', wrq, ebsize, tp);
1872                                         i--;
1873                                 }
1874                         }
1875                         bpt->b_datap->db_type = M_DATA;
1876                         ldterm_msg_upstream(q, tp);
1877                         if (!canputnext(q)) {
1878                                 bpt = NULL;
1879                                 *dofreep = 0;
1880                         } else {
1881                                 bpt = newmsg(tp);
1882                                 *dofreep = 1;
1883                         }
1884                         goto out;
1885                 }
1886         }

1888 escaped:
1889         /*
1890          * First, make sure we can fit one WHOLE multi-byte char in the
1891          * buffer.  This is one place where we have overhead even if
1892          * not in multi-byte mode; the overhead is subtracting
1893          * tp->t_maxeuc from MAX_CANON before checking.
1894          *
1895          * Allows MAX_CANON bytes in the buffer before throwing awaying
1896          * the the overflow of characters.
1897          */
1898         if ((tp->t_msglen > ((_TTY_BUFSIZ + 1) - (int)tp->t_maxeuc)) &&
1898         if ((tp->t_msglen > ((MAX_CANON + 1) - (int)tp->t_maxeuc)) &&
1899             !((tp->t_state & TS_MEUC) && tp->t_eucleft)) {
```

```
1901                 /*
1902                  * Byte will cause line to overflow, or the next EUC
1903                  * won't fit: Ring the bell or discard all input, and
1904                  * don't save the byte away.
1905                  */
1906                 if (tp->t_modes.c_iflag & IMAXBEL) {
1907                         if (canputnext(wrq))
1908                                 ldterm_outchar(CTRL('g'), wrq, ebsize, tp);
1909                         goto out;
1910                 } else {
1911                         /*
1912                          * MAX_CANON processing. free everything in
1913                          * the current line and start with the
1914                          * current character as the first character.
1915                          */
1916                         DEBUG7(("ldterm_docanon: MAX_CANON processing\n"));
1917                         freemsg(tp->t_message);
1918                         tp->t_message = NULL;
1919                         tp->t_endmsg = NULL;
1920                         tp->t_msglen = 0;
1921                         tp->t_rocount = 0;      /* if it hasn't been type */
1922                         tp->t_rocol = 0;        /* it hasn't been echoed :-) */
1923                         if (tp->t_state & TS_MEUC) {
1924                                 ASSERT(tp->t_eucp_mp);
1925                                 tp->t_eucp = tp->t_eucp_mp->b_rptr;
1926                         }
1927                         tp->t_state &= ~TS_SLNCH;
1928                         bpt = newmsg(tp);
1929                 }
1930         }
1931         /*
1932          * Add the character to the current line.
1933          */
1934         if (bpt->b_wptr >= bpt->b_datap->db_lim) {
1935                 /*
1936                  * No more room in this mblk; save this one away, and
1937                  * allocate a new one.
1938                  */
1939                 bpt->b_datap->db_type = M_DATA;
1940                 if ((bpt = allocb(IBSIZE, BPRI_MED)) == NULL)
1941                         goto out;

1943                 /*
1944                  * Chain the new one to the end of the old one, and
1945                  * mark it as the last block in the current line.
1946                  */
1947                 tp->t_endmsg->b_cont = bpt;
1948                 tp->t_endmsg = bpt;
1949         }
1950         *bpt->b_wptr++ = c;
1951         tp->t_msglen++;         /* message length in BYTES */

1953         /*
1954          * In multi-byte mode, we have to keep track of where we are.
1955          * The first bytes of multi-byte chars get the full count for the
1956          * whole character.  We don't do any column calculations
1957          * here, but we need the information for when we do. We could
1958          * come across cases where we are getting garbage on the
1959          * line, but we're in multi-byte mode.  In that case, we may
1960          * see ASCII controls come in the middle of what should have been a
1961          * multi-byte character.  Call ldterm_eucwarn...eventually, a
1962          * warning message will be printed about it.
1963          */
1964         if (tp->t_state & TS_MEUC) {
1965                 if (tp->t_eucleft) {    /* if in a multi-byte char already */
```



```
1966                         --tp->t_eucleft;
1967                         *tp->t_eucp++ = 0;      /* is a subsequent byte */
1968                         if (c < (uchar_t)0x20)
1969                                 ldterm_eucwarn(tp);
1970                 } else { /* is the first byte of a multi-byte, or is ASCII */
1971                         if (ISASCII(c)) {
1972                                 *tp->t_eucp++ =
1973                                     tp->t_csmethods.ldterm_dispwidth(c,
1974                                     (void *)tp, tp->t_modes.c_lflag & ECHOCTL);
1975                                 tp->t_codeset = 0;
1976                         } else {
1977                                 *tp->t_eucp++ =
1978                                     tp->t_csmethods.ldterm_dispwidth(c,
1979                                     (void *)tp, tp->t_modes.c_lflag & ECHOCTL);
1980                                 tp->t_eucleft =
1981                                     tp->t_csmethods.ldterm_memwidth(c,
1982                                     (void *)tp) - 1;
1983                                 tp->t_codeset = ldterm_codeset(
1984                                     tp->t_csdata.codeset_type, c);
1985                         }
1986                 }
1987         }
1988         /*
1989          * AT&T is concerned about the following but we aren't since
1990          * we have already shipped code that works.
1991          *
1992          * EOL2/XCASE should be conditioned with IEXTEN to be truly
1993          * POSIX conformant. This is going to cause problems for
1994          * pre-SVR4.0 programs that don't know about IEXTEN. Hence
1995          * EOL2/IEXTEN is not conditioned with IEXTEN.
1996          */
1997         if (!(tp->t_state & TS_SLNCH) &&
1998             (c == '\n' || (c != '\0' && (c == tp->t_modes.c_cc[VEOL] ||
1999             (c == tp->t_modes.c_cc[VEOL2]))))) {
2000                 /*
2001                  * || ((tp->t_modes.c_lflag & IEXTEN) && c ==
2002                  * tp->t_modes.c_cc[VEOL2]))))) {
2003                  */
2004                 /*
2005                  * It's a line-termination character; send the line
2006                  * up the stream.
2007                  */
2008                 bpt->b_datap->db_type = M_DATA;
2009                 ldterm_msg_upstream(q, tp);
2010                 if (tp->t_state & TS_MEUC) {
2011                         ASSERT(tp->t_eucp_mp);
2012                         tp->t_eucp = tp->t_eucp_mp->b_rptr;
2013                 }
2014                 if ((bpt = newmsg(tp)) == NULL)
2015                         goto out;
2016         } else {
2017                 /*
2018                  * Character was escaped with LNEXT.
2019                  */
2020                 if (tp->t_rocount++ == 0)
2021                         tp->t_rocol = tp->t_col;
2022                 tp->t_state &= ~(TS_SLNCH|TS_QUOT);
2023                 /*
2024                  * If the current character is a single byte and single
2025                  * column character and it is the backslash character and
2026                  * IEXTEN, then the state will have TS_QUOT.
2027                  */
2028                 if ((c == '\\') && (tp->t_modes.c_lflag & IEXTEN) &&
2029                     (!(tp->t_state & TS_MEUC) ||
2030                     ((tp->t_state & TS_MEUC) && (!tp->t_eucleft))))
2031                         tp->t_state |= TS_QUOT;
```

```
2032         }

2034         /*
2035          * Echo it.
2036          */
2037         if (tp->t_state & TS_ERASE) {
2038                 tp->t_state &= ~TS_ERASE;
2039                 if (tp->t_modes.c_lflag & ECHO)
2040                         ldterm_outchar('/', wrq, ebsize, tp);
2041         }
2042         if (tp->t_modes.c_lflag & ECHO)
2043                 (void) ldterm_echo(c, wrq, ebsize, tp);
2044         else {
2045                 /*
2046                  * Echo NL when ECHO turned off, if ECHONL flag is
2047                  * set.
2048                  */
2049                 if (c == '\n' && (tp->t_modes.c_lflag & ECHONL))
2050                         ldterm_outchar(c, wrq, ebsize, tp);
2051         }

2053 out:

2055         return (bpt);
2056 }
_____unchanged_portion_omitted_


4079 /*
4080  * Called when an M_IOCTL message is seen on the write queue; does
4081  * whatever we're supposed to do with it, and either replies
4082  * immediately or passes it to the next module down.
4083  */
4084 static void
4085 ldterm_do_ioctl(queue_t *q, mblk_t *mp)
4086 {
4087         ldtermstd_state_t *tp;
4088         struct iocblk *iocp;
4089         struct eucioc *euciocp; /* needed for EUC ioctls */
4090         ldterm_cs_data_user_t *csdp;
4091         int i;
4092         int locale_name_sz;
4093         uchar_t maxbytelen;
4094         uchar_t maxscreenlen;
4095         int error;

4097         iocp = (struct iocblk *)mp->b_rptr;
4098         tp = (ldtermstd_state_t *)q->q_ptr;

4100         switch (iocp->ioc_cmd) {

4102         case TCSETS:
4103         case TCSETSW:
4104         case TCSETSF:
4105                 {
4106                         /*
4107                          * Set current parameters and special
4108                          * characters.
4109                          */
4110                         struct termios *cb;
4111                         struct termios oldmodes;

4113                         error = miocpullup(mp, sizeof (struct termios));
4114                         if (error != 0) {
4115                                 miocnak(q, mp, 0, error);
4116                                 return;
```



```
4117                         }

4119                         cb = (struct termios *)mp->b_cont->b_rptr;

4121                         oldmodes = tp->t_amodes;
4122                         tp->t_amodes = *cb;
4123                         if ((tp->t_amodes.c_lflag & PENDIN) &&
4124                             (tp->t_modes.c_lflag & IEXTEN)) {
4125                                 /*
4126                                  * Yuk.  The C shell file completion
4127                                  * code actually uses this "feature",
4128                                  * so we have to support it.
4129                                  */
4130                                 if (tp->t_message != NULL) {
4131                                         tp->t_state |= TS_RESCAN;
4132                                         qenable(RD(q));
4133                                 }
4134                                 tp->t_amodes.c_lflag &= ~PENDIN;
4135                         }
4136                         bcopy(tp->t_amodes.c_cc, tp->t_modes.c_cc, NCCS);

4138                         /*
4139                          * ldterm_adjust_modes does not deal with
4140                          * cflags
4141                          */
4142                         tp->t_modes.c_cflag = tp->t_amodes.c_cflag;

4144                         ldterm_adjust_modes(tp);
4145                         if (chgstropts(&oldmodes, tp, RD(q)) == (-1)) {
4146                                 miocnak(q, mp, 0, EAGAIN);
4147                                 return;
4148                         }
4149                         /*
4150                          * The driver may want to know about the
4151                          * following iflags: IGNBRK, BRKINT, IGNPAR,
4152                          * PARMRK, INPCK, IXON, IXANY.
4153                          */
4154                         break;
4155                 }

4157         case TCSETA:
4158         case TCSETAW:
4159         case TCSETAF:
4160                 {
4161                         /*
4162                          * Old-style "ioctl" to set current
4163                          * parameters and special characters. Don't
4164                          * clear out the unset portions, leave them
4165                          * as they are.
4166                          */
4167                         struct termio *cb;
4168                         struct termios oldmodes;

4170                         error = miocpullup(mp, sizeof (struct termio));
4171                         if (error != 0) {
4172                                 miocnak(q, mp, 0, error);
4173                                 return;
4174                         }

4176                         cb = (struct termio *)mp->b_cont->b_rptr;

4178                         oldmodes = tp->t_amodes;
4179                         tp->t_amodes.c_iflag =
4180                             (tp->t_amodes.c_iflag & 0xffff0000 | cb->c_iflag);
4181                         tp->t_amodes.c_oflag =
4182                             (tp->t_amodes.c_oflag & 0xffff0000 | cb->c_oflag);
```

```
4183                        tp->t_amodes.c_cflag =
4184                            (tp->t_amodes.c_cflag & 0xffff0000 | cb->c_cflag);
4185                        tp->t_amodes.c_lflag =
4186                            (tp->t_amodes.c_lflag & 0xffff0000 | cb->c_lflag);
4188                        bcopy(cb->c_cc, tp->t_modes.c_cc, NCC);
4189                        /* TCGETS returns amodes, so update that too */
4190                        bcopy(cb->c_cc, tp->t_amodes.c_cc, NCC);
4192                        /* ldterm_adjust_modes does not deal with cflags */
4194                        tp->t_modes.c_cflag = tp->t_amodes.c_cflag;
4196                        ldterm_adjust_modes(tp);
4197                        if (chgstropts(&oldmodes, tp, RD(q)) == (-1)) {
4198                                miocnak(q, mp, 0, EAGAIN);
4199                                return;
4200                        }
4201                        /*
4202                         * The driver may want to know about the
4203                         * following iflags: IGNBRK, BRKINT, IGNPAR,
4204                         * PARMRK, INPCK, IXON, IXANY.
4205                         */
4206                        break;
4207                }
4209        case TCFLSH:
4210                /*
4211                 * Do the flush on the write queue immediately, and
4212                 * queue up any flush on the read queue for the
4213                 * service procedure to see.  Then turn it into the
4214                 * appropriate M_FLUSH message, so that the module
4215                 * below us doesn't have to know about TCFLSH.
4216                 */
4217                error = miocpullup(mp, sizeof (int));
4218                if (error != 0) {
4219                        miocnak(q, mp, 0, error);
4220                        return;
4221                }
4223                ASSERT(mp->b_datap != NULL);
4224                if (*(int *)mp->b_cont->b_rptr == 0) {
4225                        ASSERT(mp->b_datap != NULL);
4226                        (void) putnextctl1(q, M_FLUSH, FLUSHR);
4227                        (void) putctl1(RD(q), M_FLUSH, FLUSHR);
4228                } else if (*(int *)mp->b_cont->b_rptr == 1) {
4229                        flushq(q, FLUSHDATA);
4230                        ASSERT(mp->b_datap != NULL);
4231                        tp->t_state |= TS_FLUSHWAIT;
4232                        (void) putnextctl1(RD(q), M_FLUSH, FLUSHW);
4233                        (void) putnextctl1(q, M_FLUSH, FLUSHW);
4234                } else if (*(int *)mp->b_cont->b_rptr == 2) {
4235                        flushq(q, FLUSHDATA);
4236                        ASSERT(mp->b_datap != NULL);
4237                        (void) putnextctl1(q, M_FLUSH, FLUSHRW);
4238                        tp->t_state |= TS_FLUSHWAIT;
4239                        (void) putnextctl1(RD(q), M_FLUSH, FLUSHRW);
4240                } else {
4241                        miocnak(q, mp, 0, EINVAL);
4242                        return;
4243                }
4244                ASSERT(mp->b_datap != NULL);
4245                iocp->ioc_rval = 0;
4246                miocack(q, mp, 0, 0);
4247                return;
```



```
4249        case TCXONC:
4250                error = miocpullup(mp, sizeof (int));
4251                if (error != 0) {
4252                        miocnak(q, mp, 0, error);
4253                        return;
4254                }
4256                switch (*(int *)mp->b_cont->b_rptr) {
4257                case 0:
4258                        if (!(tp->t_state & TS_TTSTOP)) {
4259                                (void) putnextctl(q, M_STOP);
4260                                tp->t_state |= (TS_TTSTOP|TS_OFBLOCK);
4261                        }
4262                        break;
4264                case 1:
4265                        if (tp->t_state & TS_TTSTOP) {
4266                                (void) putnextctl(q, M_START);
4267                                tp->t_state &= ~(TS_TTSTOP|TS_OFBLOCK);
4268                        }
4269                        break;
4271                case 2:
4272                        (void) putnextctl(q, M_STOPI);
4273                        tp->t_state |= (TS_TBLOCK|TS_IFBLOCK);
4274                        break;
4276                case 3:
4277                        (void) putnextctl(q, M_STARTI);
4278                        tp->t_state &= ~(TS_TBLOCK|TS_IFBLOCK);
4279                        break;
4281                default:
4282                        miocnak(q, mp, 0, EINVAL);
4283                        return;
4284                }
4285                ASSERT(mp->b_datap != NULL);
4286                iocp->ioc_rval = 0;
4287                miocack(q, mp, 0, 0);
4288                return;
4289                /*
4290                 * TCSBRK is expected to be handled by the driver.
4291                 * The reason its left for the driver is that when
4292                 * the argument to TCSBRK is zero driver has to drain
4293                 * the data and sending a M_IOCACK from LDTERM before
4294                 * the driver drains the data is going to cause
4295                 * problems.
4296                 */
4298                /*
4299                 * The following are EUC related ioctls.  For
4300                 * EUC_WSET, we have to pass the information on, even
4301                 * though we ACK the call.  It's vital in the EUC
4302                 * environment that everybody downstream knows about
4303                 * the EUC codeset widths currently in use; we
4304                 * therefore pass down the information in an M_CTL
4305                 * message.  It will bottom out in the driver.
4306                 */
4307        case EUC_WSET:
4308                {
4310                        /* only needed for EUC_WSET */
4311                        struct iocblk *riocp;
4313                        mblk_t *dmp, *dmp_cont;
```

```
4315                    /*
4316                     * If the user didn't supply any information,
4317                     * NAK it.
4318                     */
4319                    error = miocpullup(mp, sizeof (struct eucioc));
4320                    if (error != 0) {
4321                            miocnak(q, mp, 0, error);
4322                            return;
4323                    }

4325                    euciocp = (struct eucioc *)mp->b_cont->b_rptr;
4326                    /*
4327                     * Check here for something reasonable.  If
4328                     * anything will take more than EUC_MAXW
4329                     * columns or more than EUC_MAXW bytes
4330                     * following SS2 or SS3, then just reject it
4331                     * out of hand. It's not impossible for us to
4332                     * do it, it just isn't reasonable.  So far,
4333                     * in the world, we've seen the absolute max
4334                     * columns to be 2 and the max number of
4335                     * bytes to be 3.  This allows room for some
4336                     * expansion of that, but it probably won't
4337                     * even be necessary. At the moment, we
4338                     * return a "range" error.  If you really
4339                     * need to, you can push EUC_MAXW up to over
4340                     * 200; it doesn't make sense, though, with
4341                     * only a CANBSIZ sized input limit (usually
4342                     * 256)!
4343                     */
4344                    for (i = 0; i < 4; i++) {
4345                            if ((euciocp->eucw[i] > EUC_MAXW) ||
4346                                (euciocp->scrw[i] > EUC_MAXW)) {
4347                                    miocnak(q, mp, 0, ERANGE);
4348                                    return;
4349                            }
4350                    }
4351                    /*
4352                     * Otherwise, save the information in tp,
4353                     * force codeset 0 (ASCII) to be one byte,
4354                     * one column.
4355                     */
4356                    cp_eucwioc(euciocp, &tp->eucwioc, EUCIN);
4357                    tp->eucwioc.eucw[0] = tp->eucwioc.scrw[0] = 1;
4358                    /*
4359                     * Now, check out whether we're doing
4360                     * multibyte processing. if we are, we need
4361                     * to allocate a block to hold the parallel
4362                     * array. By convention, we've been passed
4363                     * what amounts to a CSWIDTH definition.  We
4364                     * actually NEED the number of bytes for
4365                     * Codesets 2 & 3.
4366                     */
4367                    tp->t_maxeuc = 0;       /* reset to say we're NOT */

4369                    tp->t_state &= ~TS_MEUC;
4370                    /*
4371                     * We'll set TS_MEUC if we're doing
4372                     * multi-column OR multi- byte OR both.  It
4373                     * makes things easier...  NOTE:  If we fail
4374                     * to get the buffer we need to hold display
4375                     * widths, then DON'T let the TS_MEUC bit get
4376                     * set!
4377                     */
4378                    for (i = 0; i < 4; i++) {
4379                            if (tp->eucwioc.eucw[i] > tp->t_maxeuc)
4380                                    tp->t_maxeuc = tp->eucwioc.eucw[i];
```



```
4381                            if (tp->eucwioc.scrw[i] > 1)
4382                                    tp->t_state |= TS_MEUC;
4383                    }
4384                    if ((tp->t_maxeuc > 1) || (tp->t_state & TS_MEUC)) {
4385                            if (!tp->t_eucp_mp) {
4386                                    if ((tp->t_eucp_mp = allocb(_TTY_BUFSIZ,
4387                                        BPRI_HI)) == NULL) {
4386                                    if (!(tp->t_eucp_mp = allocb(CANBSIZ,
4387                                        BPRI_HI))) {
4388                                            tp->t_maxeuc = 1;
4389                                            tp->t_state &= ~TS_MEUC;
4390                                            cmn_err(CE_WARN,
4391                                                "Can't allocate eucp_mp");
4392                                            miocnak(q, mp, 0, ENOSR);
4393                                            return;
4394                                    }
4395                                    /*
4396                                     * here, if there's junk in
4397                                     * the canonical buffer, then
4398                                     * move the eucp pointer past
4399                                     * it, so we don't run off
4400                                     * the beginning.  This is a
4401                                     * total botch, but will
4402                                     * hopefully keep stuff from
4403                                     * getting too messed up
4404                                     * until the user flushes
4405                                     * this line!
4406                                     */
4407                                    if (tp->t_msglen) {
4408                                            tp->t_eucp =
4409                                                tp->t_eucp_mp->b_rptr;
4410                                            for (i = tp->t_msglen; i; i--)
4411                                                    *tp->t_eucp++ = 1;
4412                                    } else {
4413                                            tp->t_eucp =
4414                                                tp->t_eucp_mp->b_rptr;
4415                                    }
4416                            }
4417                            /* doing multi-byte handling */
4418                            tp->t_state |= TS_MEUC;

4420                    } else if (tp->t_eucp_mp) {
4421                            freemsg(tp->t_eucp_mp);
4422                            tp->t_eucp_mp = NULL;
4423                            tp->t_eucp = NULL;
4424                    }

4426                    /*
4427                     * Save the EUC width data we have at
4428                     * the t_csdata, set t_csdata.codeset_type to
4429                     * EUC one, and, switch the codeset methods at
4430                     * t_csmethods.
4431                     */
4432                    bzero(&tp->t_csdata.eucpc_data,
4433                        (sizeof (ldterm_eucpc_data_t) *
4434                        LDTERM_CS_MAX_CODESETS));
4435                    tp->t_csdata.eucpc_data[0].byte_length =
4436                        tp->eucwioc.eucw[1];
4437                    tp->t_csdata.eucpc_data[0].screen_width =
4438                        tp->eucwioc.scrw[1];
4439                    tp->t_csdata.eucpc_data[1].byte_length =
4440                        tp->eucwioc.eucw[2];
4441                    tp->t_csdata.eucpc_data[1].screen_width =
4442                        tp->eucwioc.scrw[2];
4443                    tp->t_csdata.eucpc_data[2].byte_length =
4444                        tp->eucwioc.eucw[3];
```

```
4445                        tp->t_csdata.eucpc_data[2].screen_width =
4446                            tp->eucwioc.scrw[3];
4447                        tp->t_csdata.version = LDTERM_DATA_VERSION;
4448                        tp->t_csdata.codeset_type = LDTERM_CS_TYPE_EUC;
4449                        /*
4450                         * We are not using the 'csinfo_num' anyway if the
4451                         * current codeset type is EUC. So, set it to
4452                         * the maximum possible.
4453                         */
4454                        tp->t_csdata.csinfo_num =
4455                            LDTERM_CS_TYPE_EUC_MAX_SUBCS;
4456                        if (tp->t_csdata.locale_name != (char *)NULL) {
4457                                kmem_free(tp->t_csdata.locale_name,
4458                                    strlen(tp->t_csdata.locale_name) + 1);
4459                                tp->t_csdata.locale_name = (char *)NULL;
4460                        }
4461                        tp->t_csmethods = cs_methods[LDTERM_CS_TYPE_EUC];

4463                        /*
4464                         * If we are able to allocate two blocks (the
4465                         * iocblk and the associated data), then pass
4466                         * it downstream, otherwise we'll need to NAK
4467                         * it, and drop whatever we WERE able to
4468                         * allocate.
4469                         */
4470                        if ((dmp = mkiocb(EUC_WSET)) == NULL) {
4471                                miocnak(q, mp, 0, ENOSR);
4472                                return;
4473                        }
4474                        if ((dmp_cont = allocb(EUCSIZE, BPRI_HI)) == NULL) {
4475                                freemsg(dmp);
4476                                miocnak(q, mp, 0, ENOSR);
4477                                return;
4478                        }

4480                        /*
4481                         * We got both buffers.  Copy out the EUC
4482                         * information (as we received it, not what
4483                         * we're using!) & pass it on.
4484                         */
4485                        bcopy(mp->b_cont->b_rptr, dmp_cont->b_rptr, EUCSIZE);
4486                        dmp_cont->b_wptr += EUCSIZE;
4487                        dmp->b_cont = dmp_cont;
4488                        dmp->b_datap->db_type = M_CTL;
4489                        dmp_cont->b_datap->db_type = M_DATA;
4490                        riocp = (struct iocblk *)dmp->b_rptr;
4491                        riocp->ioc_count = EUCSIZE;
4492                        putnext(q, dmp);

4494                        /*
4495                         * Now ACK the ioctl.
4496                         */
4497                        iocp->ioc_rval = 0;
4498                        miocack(q, mp, 0, 0);
4499                        return;
4500                }

4502        case EUC_WGET:
4503                error = miocpullup(mp, sizeof (struct eucioc));
4504                if (error != 0) {
4505                        miocnak(q, mp, 0, error);
4506                        return;
4507                }
4508                euciocp = (struct eucioc *)mp->b_cont->b_rptr;
4509                cp_eucwioc(&tp->eucwioc, euciocp, EUCOUT);
4510                iocp->ioc_rval = 0;
```



```
4511                miocack(q, mp, EUCSIZE, 0);
4512                return;

4514        case CSDATA_SET:
4515                error = miocpullup(mp, sizeof (ldterm_cs_data_user_t));
4516                if (error != 0) {
4517                        miocnak(q, mp, 0, error);
4518                        return;
4519                }

4521                csdp = (ldterm_cs_data_user_t *)mp->b_cont->b_rptr;

4523                /* Validate the codeset data provided. */
4524                if (csdp->version > LDTERM_DATA_VERSION ||
4525                    csdp->codeset_type < LDTERM_CS_TYPE_MIN ||
4526                    csdp->codeset_type > LDTERM_CS_TYPE_MAX) {
4527                        miocnak(q, mp, 0, ERANGE);
4528                        return;
4529                }

4531                if ((csdp->codeset_type == LDTERM_CS_TYPE_EUC &&
4532                    csdp->csinfo_num > LDTERM_CS_TYPE_EUC_MAX_SUBCS) ||
4533                    (csdp->codeset_type == LDTERM_CS_TYPE_PCCS &&
4534                    (csdp->csinfo_num < LDTERM_CS_TYPE_PCCS_MIN_SUBCS ||
4535                    csdp->csinfo_num > LDTERM_CS_TYPE_PCCS_MAX_SUBCS))) {
4536                        miocnak(q, mp, 0, ERANGE);
4537                        return;
4538                }

4540                maxbytelen = maxscreenlen = 0;
4541                if (csdp->codeset_type == LDTERM_CS_TYPE_EUC) {
4542                        for (i = 0; i < LDTERM_CS_TYPE_EUC_MAX_SUBCS; i++) {
4543                                if (csdp->eucpc_data[i].byte_length >
4544                                    EUC_MAXW ||
4545                                    csdp->eucpc_data[i].screen_width >
4546                                    EUC_MAXW) {
4547                                        miocnak(q, mp, 0, ERANGE);
4548                                        return;
4549                                }

4551                                if (csdp->eucpc_data[i].byte_length >
4552                                    maxbytelen)
4553                                        maxbytelen =
4554                                            csdp->eucpc_data[i].byte_length;
4555                                if (csdp->eucpc_data[i].screen_width >
4556                                    maxscreenlen)
4557                                        maxscreenlen =
4558                                            csdp->eucpc_data[i].screen_width;
4559                        }
4560                        /* POSIX/C locale? */
4561                        if (maxbytelen == 0 && maxscreenlen == 0)
4562                                maxbytelen = maxscreenlen = 1;
4563                } else if (csdp->codeset_type == LDTERM_CS_TYPE_PCCS) {
4564                        for (i = 0; i < LDTERM_CS_MAX_CODESETS; i++) {
4565                                if (csdp->eucpc_data[i].byte_length >
4566                                    LDTERM_CS_MAX_BYTE_LENGTH) {
4567                                        miocnak(q, mp, 0, ERANGE);
4568                                        return;
4569                                }
4570                                if (csdp->eucpc_data[i].byte_length >
4571                                    maxbytelen)
4572                                        maxbytelen =
4573                                            csdp->eucpc_data[i].byte_length;
4574                                if (csdp->eucpc_data[i].screen_width >
4575                                    maxscreenlen)
4576                                        maxscreenlen =
```

```
4577                                        csdp->eucpc_data[i].screen_width;
4578                    }
4579            } else if (csdp->codeset_type == LDTERM_CS_TYPE_UTF8) {
4580                    maxbytelen = 4;
4581                    maxscreenlen = 2;
4582            }

4584            locale_name_sz = 0;
4585            if (csdp->locale_name) {
4586                    for (i = 0; i < MAXNAMELEN; i++)
4587                            if (csdp->locale_name[i] == '\0')
4588                                    break;
4589                    /*
4590                     * We cannot have any string that is not NULL byte
4591                     * terminated.
4592                     */
4593                    if (i >= MAXNAMELEN) {
4594                            miocnak(q, mp, 0, ERANGE);
4595                            return;
4596                    }

4598                    locale_name_sz = i + 1;
4599            }

4601            /*
4602             * As the final check, if there was invalid codeset_type
4603             * given, or invalid byte_length was specified, it's an error.
4604             */
4605            if (maxbytelen <= 0 || maxscreenlen <= 0) {
4606                    miocnak(q, mp, 0, ERANGE);
4607                    return;
4608            }

4610            /* Do the switching. */
4611            tp->t_maxeuc = maxbytelen;
4612            tp->t_state &= ~TS_MEUC;
4613            if (maxbytelen > 1 || maxscreenlen > 1) {
4614                    if (!tp->t_eucp_mp) {
4615                            if (!(tp->t_eucp_mp = allocb(_TTY_BUFSIZ,
4615                            if (!(tp->t_eucp_mp = allocb(CANBSIZ,
4616                                BPRI_HI))) {
4617                                cmn_err(CE_WARN,
4618                                    "Can't allocate eucp_mp");
4619                                miocnak(q, mp, 0, ENOSR);
4620                                return;
4621                            }
4622                            /*
4623                             * If there's junk in the canonical buffer,
4624                             * then move the eucp pointer past it,
4625                             * so we don't run off the beginning. This is
4626                             * a total botch, but will hopefully keep
4627                             * stuff from getting too messed up until
4628                             * the user flushes this line!
4629                             */
4630                            if (tp->t_msglen) {
4631                                    tp->t_eucp = tp->t_eucp_mp->b_rptr;
4632                                    for (i = tp->t_msglen; i; i--)
4633                                            *tp->t_eucp++ = 1;
4634                            } else {
4635                                    tp->t_eucp = tp->t_eucp_mp->b_rptr;
4636                            }
4637                    }

4639                    /*
4640                     * We only set TS_MEUC for a multibyte/multi-column
4641                     * codeset.
```



```
4642                     */
4643                    tp->t_state |= TS_MEUC;

4645                    tp->t_csdata.version = csdp->version;
4646                    tp->t_csdata.codeset_type = csdp->codeset_type;
4647                    tp->t_csdata.csinfo_num = csdp->csinfo_num;
4648                    bcopy(csdp->eucpc_data, tp->t_csdata.eucpc_data,
4649                        sizeof (ldterm_eucpc_data_t) *
4650                        LDTERM_CS_MAX_CODESETS);
4651                    tp->t_csmethods = cs_methods[csdp->codeset_type];

4653                    if (csdp->codeset_type == LDTERM_CS_TYPE_EUC) {
4654                            tp->eucwioc.eucw[0] = 1;
4655                            tp->eucwioc.scrw[0] = 1;

4657                            tp->eucwioc.eucw[1] =
4658                                csdp->eucpc_data[0].byte_length;
4659                            tp->eucwioc.scrw[1] =
4660                                csdp->eucpc_data[0].screen_width;

4662                            tp->eucwioc.eucw[2] =
4663                                csdp->eucpc_data[1].byte_length + 1;
4664                            tp->eucwioc.scrw[2] =
4665                                csdp->eucpc_data[1].screen_width;

4667                            tp->eucwioc.eucw[3] =
4668                                csdp->eucpc_data[2].byte_length + 1;
4669                            tp->eucwioc.scrw[3] =
4670                                csdp->eucpc_data[2].screen_width;
4671                    } else {
4672                            /*
4673                             * We are not going to use this data
4674                             * structure. So, clear it. Also, stty(1) will
4675                             * make use of the cleared tp->eucwioc when
4676                             * it prints out codeset width setting.
4677                             */
4678                            bzero(&tp->eucwioc, EUCSIZE);
4679                    }
4680            } else {
4681                    /*
4682                     * If this codeset is a single byte codeset that
4683                     * requires only single display column for all
4684                     * characters, we switch to default EUC codeset
4685                     * methods and data setting.
4686                     */

4688                    if (tp->t_eucp_mp) {
4689                            freemsg(tp->t_eucp_mp);
4690                            tp->t_eucp_mp = NULL;
4691                            tp->t_eucp = NULL;
4692                    }

4694                    bzero(&tp->eucwioc, EUCSIZE);
4695                    tp->eucwioc.eucw[0] = 1;
4696                    tp->eucwioc.scrw[0] = 1;
4697                    if (tp->t_csdata.locale_name != (char *)NULL) {
4698                            kmem_free(tp->t_csdata.locale_name,
4699                                strlen(tp->t_csdata.locale_name) + 1);
4700                    }
4701                    tp->t_csdata = default_cs_data;
4702                    tp->t_csmethods = cs_methods[LDTERM_CS_TYPE_EUC];
4703            }

4705            /* Copy over locale_name. */
4706            if (tp->t_csdata.locale_name != (char *)NULL) {
4707                    kmem_free(tp->t_csdata.locale_name,
```

```
4708                                strlen(tp->t_csdata.locale_name) + 1);
4709                }
4710                if (locale_name_sz > 1) {
4711                        tp->t_csdata.locale_name = (char *)kmem_alloc(
4712                            locale_name_sz, KM_SLEEP);
4713                        (void) strcpy(tp->t_csdata.locale_name,
4714                            csdp->locale_name);
4715                } else {
4716                        tp->t_csdata.locale_name = (char *)NULL;
4717                }

4719                /*
4720                 * Now ACK the ioctl.
4721                 */
4722                iocp->ioc_rval = 0;
4723                miocack(q, mp, 0, 0);
4724                return;

4726        case CSDATA_GET:
4727                error = miocpullup(mp, sizeof (ldterm_cs_data_user_t));
4728                if (error != 0) {
4729                        miocnak(q, mp, 0, error);
4730                        return;
4731                }

4733                csdp = (ldterm_cs_data_user_t *)mp->b_cont->b_rptr;

4735                csdp->version = tp->t_csdata.version;
4736                csdp->codeset_type = tp->t_csdata.codeset_type;
4737                csdp->csinfo_num = tp->t_csdata.csinfo_num;
4738                csdp->pad = tp->t_csdata.pad;
4739                if (tp->t_csdata.locale_name) {
4740                        (void) strcpy(csdp->locale_name,
4741                            tp->t_csdata.locale_name);
4742                } else {
4743                        csdp->locale_name[0] = '\0';
4744                }
4745                bcopy(tp->t_csdata.eucpc_data, csdp->eucpc_data,
4746                    sizeof (ldterm_eucpc_data_t) * LDTERM_CS_MAX_CODESETS);
4747                /*
4748                 * If the codeset is an EUC codeset and if it has 2nd and/or
4749                 * 3rd supplementary codesets, we subtract one from each
4750                 * byte length of the supplementary codesets. This is
4751                 * because single shift characters, SS2 and SS3, are not
4752                 * included in the byte lengths in the user space.
4753                 */
4754                if (csdp->codeset_type == LDTERM_CS_TYPE_EUC) {
4755                        if (csdp->eucpc_data[1].byte_length)
4756                                csdp->eucpc_data[1].byte_length -= 1;
4757                        if (csdp->eucpc_data[2].byte_length)
4758                                csdp->eucpc_data[2].byte_length -= 1;
4759                }
4760                iocp->ioc_rval = 0;
4761                miocack(q, mp, sizeof (ldterm_cs_data_user_t), 0);
4762                return;

4764        case PTSSTTY:
4765                tp->t_state |= TS_ISPTSTTY;
4766                break;

4768        }

4770        putnext(q, mp);
4771 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   22641 Thu Jul 27 16:35:43 2017
new/usr/src/uts/common/io/ptem.c
8527 tty buffer/queue sizes should be larger
**********************************************************
```

```
   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 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
  23 /*        All Rights Reserved   */


  26 /*
  27  * Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
  28  * Use is subject to license terms.
  29  */

  31 #pragma ident   "%Z%%M% %I%     %E% SMI" /* from S5R4 1.13 */

  31 /*
  32  * Description:
  33  *
  34  * The PTEM streams module is used as a pseudo driver emulator.  Its purpose
  35  * is to emulate the ioctl() functions of a terminal device driver.
  36  */

  38 #include <sys/types.h>
  39 #include <sys/param.h>
  40 #include <sys/stream.h>
  41 #include <sys/stropts.h>
  42 #include <sys/strsun.h>
  43 #include <sys/termio.h>
  44 #include <sys/pcb.h>
  45 #include <sys/signal.h>
  46 #include <sys/cred.h>
  47 #include <sys/strtty.h>
  48 #include <sys/errno.h>
  49 #include <sys/cmn_err.h>
  50 #include <sys/jioctl.h>
  51 #include <sys/ptem.h>
  52 #include <sys/ptms.h>
  53 #include <sys/debug.h>
  54 #include <sys/kmem.h>
  55 #include <sys/ddi.h>
  56 #include <sys/sunddi.h>
  57 #include <sys/conf.h>
  58 #include <sys/modctl.h>
```



```
  60 extern struct streamtab pteminfo;

  62 static struct fmodsw fsw = {
  63         "ptem",
  64         &pteminfo,
  65         D_MTQPAIR | D_MP
  66 };
_____unchanged_portion_omitted_

  94 /*
  95  * stream data structure definitions
  96  */
  97 static int ptemopen(queue_t *, dev_t  *, int, int, cred_t *);
  98 static int ptemclose(queue_t *, int, cred_t *);
  99 static void ptemrput(queue_t *, mblk_t *);
 100 static void ptemwput(queue_t *, mblk_t *);
 101 static void ptemwsrv(queue_t *);

 103 static struct module_info ptem_info = {
 104         0xabcd,
 105         "ptem",
 106         0,
 107         _TTY_BUFSIZ,
 108         _TTY_BUFSIZ,
 109         512,
 110         512,
 109         128
 110 };
_____unchanged_portion_omitted_

 139 static void     ptioc(queue_t *, mblk_t *, int);
 140 static int      ptemwmsg(queue_t *, mblk_t *);

 142 /*
 143  * ptemopen - open routine gets called when the module gets pushed onto the
 144  * stream.
 145  */
 146 /* ARGSUSED */
 147 static int
 148 ptemopen(
 149         queue_t    *q,          /* pointer to the read side queue */
 150         dev_t   *devp,          /* pointer to stream tail's dev */
 151         int     oflag,          /* the user open(2) supplied flags */
 152         int     sflag,          /* open state flag */
 153         cred_t *credp)          /* credentials */
 154 {
 155         struct ptem *ntp;       /* ptem entry for this PTEM module */
 156         mblk_t *mop;            /* an setopts mblk */
 157         struct stroptions *sop;
 158         struct termios *termiosp;
 159         int len;

 161         if (sflag != MODOPEN)
 162                 return (EINVAL);

 164         if (q->q_ptr != NULL) {
 165                 /* It's already attached. */
 166                 return (0);
 167         }

 169         /*
 170          * Allocate state structure.
 171          */
 172         ntp = kmem_alloc(sizeof (*ntp), KM_SLEEP);

 174         /*
```

```
175          * Allocate a message block, used to pass the zero length message for
176          * "stty 0".
177          *
178          * NOTE: it's better to find out if such a message block can be
179          *       allocated before it's needed than to not be able to
180          *       deliver (for possible lack of buffers) when a hang-up
181          *       occurs.
182          */
183         if ((ntp->dack_ptr = allocb(4, BPRI_MED)) == NULL) {
184                 kmem_free(ntp, sizeof (*ntp));
185                 return (EAGAIN);
186         }
188         /*
189          * Initialize an M_SETOPTS message to set up hi/lo water marks on
190          * stream head read queue and add controlling tty if not set.
191          */
192         mop = allocb(sizeof (struct stroptions), BPRI_MED);
193         if (mop == NULL) {
194                 freemsg(ntp->dack_ptr);
195                 kmem_free(ntp, sizeof (*ntp));
196                 return (EAGAIN);
197         }
198         mop->b_datap->db_type = M_SETOPTS;
199         mop->b_wptr += sizeof (struct stroptions);
200         sop = (struct stroptions *)mop->b_rptr;
201         sop->so_flags = SO_HIWAT | SO_LOWAT | SO_ISTTY;
202         sop->so_hiwat = _TTY_BUFSIZ;
204         sop->so_hiwat = 512;
203         sop->so_lowat = 256;
205         /*
206          * Cross-link.
207          */
208         ntp->q_ptr = q;
209         q->q_ptr = ntp;
210         WR(q)->q_ptr = ntp;
212         /*
213          * Get termios defaults.  These are stored as
214          * a property in the "options" node.
215          */
216         if (ddi_getlongprop(DDI_DEV_T_ANY, ddi_root_node(), 0, "ttymodes",
217             (caddr_t)&termiosp, &len) == DDI_PROP_SUCCESS &&
218             len == sizeof (struct termios)) {
220                 ntp->cflags = termiosp->c_cflag;
221                 kmem_free(termiosp, len);
222         } else {
223                 /*
224                  * Gack!  Whine about it.
225                  */
226                 cmn_err(CE_WARN, "ptem: Couldn't get ttymodes property!");
227         }
228         ntp->wsz.ws_row = 0;
229         ntp->wsz.ws_col = 0;
230         ntp->wsz.ws_xpixel = 0;
231         ntp->wsz.ws_ypixel = 0;
233         ntp->state = 0;
235         /*
236          * Commit to the open and send the M_SETOPTS off to the stream head.
237          */
238         qprocson(q);
239         putnext(q, mop);
```



```
241         return (0);
242 }
_____unchanged_portion_omitted_
429 /*
430  * ptemwput - Module write queue put procedure.
431  *
432  * This is called from the module or stream head upstream.
433  *
434  * XXX: This routine is quite lazy about handling allocation failures,
435  *      basically just giving up and reporting failure.  It really ought to
436  *      set up bufcalls and only fail when it's absolutely necessary.
437  */
438 static void
439 ptemwput(queue_t *q, mblk_t *mp)
440 {
441         struct ptem *ntp = (struct ptem *)q->q_ptr;
442         struct iocblk *iocp;    /* outgoing ioctl structure */
443         struct copyresp *resp;
444         unsigned char type = mp->b_datap->db_type;
446         if (type >= QPCTL) {
447                 switch (type) {
449                 case M_IOCDATA:
450                         resp = (struct copyresp *)mp->b_rptr;
451                         if (resp->cp_rval) {
452                                 /*
453                                  * Just free message on failure.
454                                  */
455                                 freemsg(mp);
456                                 break;
457                         }
459                         /*
460                          * Only need to copy data for the SET case.
461                          */
462                         switch (resp->cp_cmd) {
464                                 case TIOCSWINSZ:
465                                         ptioc(q, mp, WRSIDE);
466                                         break;
468                                 case JWINSIZE:
469                                 case TIOCGWINSZ:
470                                         mioc2ack(mp, NULL, 0, 0);
471                                         qreply(q, mp);
472                                         break;
474                                 default:
475                                         freemsg(mp);
476                         }
477                         break;
479                 case M_FLUSH:
480                         if (*mp->b_rptr & FLUSHW) {
481                                 if ((ntp->state & IS_PTSTTY) &&
482                                     (*mp->b_rptr & FLUSHBAND))
483                                         flushband(q, *(mp->b_rptr + 1),
484                                             FLUSHDATA);
485                                 flushband(q, *(mp->b_rptr + 1), FLUSHDATA);
485                                 else
486                                         flushq(q, FLUSHDATA);
487                         }
```

```
488                         putnext(q, mp);
489                         break;

491                 case M_READ:
492                         freemsg(mp);
493                         break;

495                 case M_STOP:
496                         /*
497                          * Set the output flow control state.
498                          */
499                         ntp->state |= OFLOW_CTL;
500                         putnext(q, mp);
501                         break;

503                 case M_START:
504                         /*
505                          * Relieve the output flow control state.
506                          */
507                         ntp->state &= ~OFLOW_CTL;
508                         putnext(q, mp);
509                         qenable(q);
510                         break;
511                 default:
512                         putnext(q, mp);
513                         break;
514                 }
515                 return;
516         }
517         /*
518          * If our queue is nonempty or flow control persists
519          * downstream or module in stopped state, queue this message.
520          */
521         if (q->q_first != NULL || !bcanputnext(q, mp->b_band)) {
522                 /*
523                  * Exception: ioctls, except for those defined to
524                  * take effect after output has drained, should be
525                  * processed immediately.
526                  */
527                 switch (type) {

529                 case M_IOCTL:
530                         iocp = (struct iocblk *)mp->b_rptr;
531                         switch (iocp->ioc_cmd) {
532                         /*
533                          * Queue these.
534                          */
535                         case TCSETSW:
536                         case TCSETSF:
537                         case TCSETAW:
538                         case TCSETAF:
539                         case TCSBRK:
540                                 break;

542                         /*
543                          * Handle all others immediately.
544                          */
545                         default:
546                                 (void) ptemwmsg(q, mp);
547                                 return;
548                         }
549                         break;

551                 case M_DELAY: /* tty delays not supported */
552                         freemsg(mp);
553                         return;
```



```
555                 case M_DATA:
556                         if ((mp->b_wptr - mp->b_rptr) < 0) {
557                                 /*
558                                  * Free all bad length messages.
559                                  */
560                                 freemsg(mp);
561                                 return;
562                         } else if ((mp->b_wptr - mp->b_rptr) == 0) {
563                                 if (!(ntp->state & IS_PTSTTY)) {
564                                         freemsg(mp);
565                                         return;
566                                 }
567                         }
568                 }
569                 (void) putq(q, mp);
570                 return;
571         }
572         /*
573          * fast path into ptemwmsg to dispose of mp.
574          */
575         if (!ptemwmsg(q, mp))
576                 (void) putq(q, mp);
577 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   18430 Thu Jul 27 16:35:45 2017
new/usr/src/uts/common/io/ptm.c
8527 tty buffer/queue sizes should be larger
**********************************************************
______unchanged_portion_omitted_


 301 /* ARGSUSED */
 302 /*
 303  * Open a minor of the master device. Store the write queue pointer and set the
 304  * pt_state field to (PTMOPEN | PTLOCK).
 305  * This code will work properly with both clone opens and direct opens of the
 306  * master device.
 307  */
 308 static int
 309 ptmopen(
 310         queue_t *rqp,           /* pointer to the read side queue */
 311         dev_t   *devp,          /* pointer to stream tail's dev */
 312         int     oflag,          /* the user open(2) supplied flags */
 313         int     sflag,          /* open state flag */
 314         cred_t  *credp)         /* credentials */
 315 {
 316         struct pt_ttys  *ptmp;
 317         mblk_t          *mop;           /* ptr to a setopts message block */
 318         struct stroptions *sop;
 319         minor_t         dminor = getminor(*devp);

 321         /* Allow reopen */
 322         if (rqp->q_ptr != NULL)
 323                 return (0);

 325         if (sflag & MODOPEN)
 326                 return (ENXIO);

 328         if (!(sflag & CLONEOPEN) && dminor != 0) {
 329                 /*
 330                  * This is a direct open to specific master device through an
 331                  * artificially created entry with specific minor in
 332                  * /dev/directory. Such behavior is not supported.
 333                  */
 334                 return (ENXIO);
 335         }

 337         /*
 338          * The master open requires that the slave be attached
 339          * before it returns so that attempts to open the slave will
 340          * succeeed
 341          */
 342         if (ptms_attach_slave() != 0) {
 343                 return (ENXIO);
 344         }

 346         mop = allocb(sizeof (struct stroptions), BPRI_MED);
 347         if (mop == NULL) {
 348                 DDBG("ptmopen(): mop allocation failed\n", 0);
 349                 return (ENOMEM);
 350         }

 352         if ((ptmp = pt_ttys_alloc()) == NULL) {
 353                 DDBG("ptmopen(): pty allocation failed\n", 0);
 354                 freemsg(mop);
 355                 return (ENOMEM);
 356         }

 358         dminor = ptmp->pt_minor;
```



```
 360         DDBGP("ptmopen(): allocated ptmp %p\n", (uintptr_t)ptmp);
 361         DDBG("ptmopen(): allocated minor %d\n", dminor);

 363         WR(rqp)->q_ptr = rqp->q_ptr = ptmp;

 365         qprocson(rqp);

 367         /* Allow slave to send messages to master */
 368         PT_ENTER_WRITE(ptmp);
 369         ptmp->ptm_rdq = rqp;
 370         PT_EXIT_WRITE(ptmp);

 372         /*
 373          * set up hi/lo water marks on stream head read queue
 374          * and add controlling tty if not set
 375          */
 376         mop->b_datap->db_type = M_SETOPTS;
 377         mop->b_wptr += sizeof (struct stroptions);
 378         sop = (struct stroptions *)mop->b_rptr;
 379         if (oflag & FNOCTTY)
 380                 sop->so_flags = SO_HIWAT | SO_LOWAT;
 381         else
 382                 sop->so_flags = SO_HIWAT | SO_LOWAT | SO_ISTTY;
 383         sop->so_hiwat = _TTY_BUFSIZ;
 383         sop->so_hiwat = 512;
 384         sop->so_lowat = 256;
 385         putnext(rqp, mop);

 387         /*
 388          * The input, devp, is a major device number, the output is put
 389          * into the same parm as a major,minor pair.
 390          */
 391         *devp = makedevice(getmajor(*devp), dminor);

 393         return (0);
 394 }
______unchanged_portion_omitted_
```

```
**********************************************************
   18792 Thu Jul 27 16:35:46 2017
new/usr/src/uts/common/io/pts.c
8527 tty buffer/queue sizes should be larger
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */



/*
 * Pseudo Terminal Slave Driver.
 *
 * The pseudo-tty subsystem simulates a terminal connection, where the master
 * side represents the terminal and the slave represents the user process's
 * special device end point. The master device is set up as a cloned device
 * where its major device number is the major for the clone device and its minor
 * device number is the major for the ptm driver. There are no nodes in the file
 * system for master devices. The master pseudo driver is opened using the
 * open(2) system call with /dev/ptmx as the device parameter.  The clone open
 * finds the next available minor device for the ptm major device.
 *
 * A master device is available only if it and its corresponding slave device
 * are not already open. When the master device is opened, the corresponding
 * slave device is automatically locked out. Only one open is allowed on a
 * master device.  Multiple opens are allowed on the slave device.  After both
 * the master and slave have been opened, the user has two file descriptors
 * which are the end points of a full duplex connection composed of two streams
 * which are automatically connected at the master and slave drivers. The user
 * may then push modules onto either side of the stream pair.
 *
 * The master and slave drivers pass all messages to their adjacent queues.
 * Only the M_FLUSH needs some processing.  Because the read queue of one side
 * is connected to the write queue of the other, the FLUSHR flag is changed to
 * the FLUSHW flag and vice versa. When the master device is closed an M_HANGUP
 * message is sent to the slave device which will render the device
 * unusable. The process on the slave side gets the EIO when attempting to write
 * on that stream but it will be able to read any data remaining on the stream
 * head read queue.  When all the data has been read, read() returns 0
 * indicating that the stream can no longer be used.  On the last close of the
 * slave device, a 0-length message is sent to the master device. When the
 * application on the master side issues a read() or getmsg() and 0 is returned,
```



```
 * the user of the master device decides whether to issue a close() that
 * dismantles the pseudo-terminal subsystem. If the master device is not closed,
 * the pseudo-tty subsystem will be available to another user to open the slave
 * device.
 *
 * Synchronization:
 *
 *   All global data synchronization between ptm/pts is done via global
 *   ptms_lock mutex which is initialized at system boot time from
 *   ptms_initspace (called from space.c).
 *
 *   Individual fields of pt_ttys structure (except ptm_rdq, pts_rdq and
 *   pt_nullmsg) are protected by pt_ttys.pt_lock mutex.
 *
 *   PT_ENTER_READ/PT_ENTER_WRITE are reference counter based read-write locks
 *   which allow reader locks to be reacquired by the same thread (usual
 *   reader/writer locks can't be used for that purpose since it is illegal for
 *   a thread to acquire a lock it already holds, even as a reader). The sole
 *   purpose of these macros is to guarantee that the peer queue will not
 *   disappear (due to closing peer) while it is used. It is safe to use
 *   PT_ENTER_READ/PT_EXIT_READ brackets across calls like putq/putnext (since
 *   they are not real locks but reference counts).
 *
 *   PT_ENTER_WRITE/PT_EXIT_WRITE brackets are used ONLY in master/slave
 *   open/close paths to modify ptm_rdq and pts_rdq fields. These fields should
 *   be set to appropriate queues *after* qprocson() is called during open (to
 *   prevent peer from accessing the queue with incomplete plumbing) and set to
 *   NULL before qprocsoff() is called during close.
 *
 *   The pt_nullmsg field is only used in open/close routines and it is also
 *   protected by PT_ENTER_WRITE/PT_EXIT_WRITE brackets to avoid extra mutex
 *   holds.
 *
 * Lock Ordering:
 *
 *   If both ptms_lock and per-pty lock should be held, ptms_lock should always
 *   be entered first, followed by per-pty lock.
 *
 * See ptms.h, ptm.c and ptms_conf.c fore more information.
 *
 */

#include <sys/types.h>
#include <sys/param.h>
#include <sys/sysmacros.h>
#include <sys/stream.h>
#include <sys/stropts.h>
#include <sys/stat.h>
#include <sys/errno.h>
#include <sys/debug.h>
#include <sys/cmn_err.h>
#include <sys/ptms.h>
#include <sys/systm.h>
#include <sys/modctl.h>
#include <sys/conf.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/cred.h>
#include <sys/zone.h>

#ifdef DEBUG
int pts_debug = 0;
#define DBG(a)   if (pts_debug) cmn_err(CE_NOTE, a)
#else
#define DBG(a)
#endif
```

```
129 static int ptsopen(queue_t *, dev_t *, int, int, cred_t *);
130 static int ptsclose(queue_t *, int, cred_t *);
131 static void ptswput(queue_t *, mblk_t *);
132 static void ptsrsrv(queue_t *);
133 static void ptswsrv(queue_t *);

135 /*
136  * Slave Stream Pseudo Terminal Module: stream data structure definitions
137  */
138 static struct module_info pts_info = {
139         0xface,
140         "pts",
141         0,
142         _TTY_BUFSIZ,
143         _TTY_BUFSIZ,
142         512,
143         512,
144         128
145 };
```

*unchanged_portion_omitted_*

```
278 /* ARGSUSED */
279 /*
280  * Open the slave device. Reject a clone open and do not allow the
281  * driver to be pushed. If the slave/master pair is locked or if
282  * the master is not open, return EACCESS.
283  * Upon success, store the write queue pointer in private data and
284  * set the PTSOPEN bit in the pt_state field.
285  */
286 static int
287 ptsopen(
288         queue_t *rqp,           /* pointer to the read side queue */
289         dev_t   *devp,          /* pointer to stream tail's dev */
290         int     oflag,          /* the user open(2) supplied flags */
291         int     sflag,          /* open state flag */
292         cred_t  *credp)         /* credentials */
293 {
294         struct pt_ttys  *ptsp;
295         mblk_t          *mp;
296         mblk_t          *mop;   /* ptr to a setopts message block */
297         minor_t         dminor = getminor(*devp);
298         struct stroptions *sop;

300         DDBG("entering ptsopen(%d)", dminor);

302         if (sflag != 0) {
303                 return (EINVAL);
304         }

306         mutex_enter(&ptms_lock);
307         ptsp = ptms_minor2ptty(dminor);

309         if (ptsp == NULL) {
310                 mutex_exit(&ptms_lock);
311                 return (ENXIO);
312         }
313         mutex_enter(&ptsp->pt_lock);

315         /*
316          * Prevent opens from zones other than the one blessed by ptm.  We
317          * can't even allow the global zone to open all pts's, as it would
318          * otherwise inproperly be able to claim pts's already opened by zones.
319          */
320         if (ptsp->pt_zoneid != getzoneid()) {
321                 mutex_exit(&ptsp->pt_lock);
```



```
322                 mutex_exit(&ptms_lock);
323                 return (EPERM);
324         }

326         /*
327          * Allow reopen of this device.
328          */
329         if (rqp->q_ptr != NULL) {
330                 ASSERT(rqp->q_ptr == ptsp);
331                 ASSERT(ptsp->pts_rdq == rqp);
332                 mutex_exit(&ptsp->pt_lock);
333                 mutex_exit(&ptms_lock);
334                 return (0);
335         }

337         DDBGP("ptsopen: p = %p\n", (uintptr_t)ptsp);
338         DDBG("ptsopen: state = %x\n", ptsp->pt_state);


341         ASSERT(ptsp->pt_minor == dminor);

343         if ((ptsp->pt_state & PTLOCK) || !(ptsp->pt_state & PTMOPEN)) {
344                 mutex_exit(&ptsp->pt_lock);
345                 mutex_exit(&ptms_lock);
346                 return (EAGAIN);
347         }

349         /*
350          * if already, open simply return...
351          */
352         if (ptsp->pt_state & PTSOPEN) {
353                 ASSERT(rqp->q_ptr == ptsp);
354                 ASSERT(ptsp->pts_rdq == rqp);
355                 mutex_exit(&ptsp->pt_lock);
356                 mutex_exit(&ptms_lock);
357                 return (0);
358         }

360         /*
361          * Allocate message block for setting stream head options.
362          */
363         if ((mop = allocb(sizeof (struct stroptions), BPRI_MED)) == NULL) {
364                 mutex_exit(&ptsp->pt_lock);
365                 mutex_exit(&ptms_lock);
366                 return (ENOMEM);
367         }

369         /*
370          * Slave should send zero-length message to a master when it is
371          * closing. If memory is low at that time, master will not detect slave
372          * closes, this pty will not be deallocated. So, preallocate this
373          * zero-length message block early.
374          */
375         if ((mp = allocb(0, BPRI_MED)) == NULL) {
376                 mutex_exit(&ptsp->pt_lock);
377                 mutex_exit(&ptms_lock);
378                 freemsg(mop);
379                 return (ENOMEM);
380         }

382         ptsp->pt_state |= PTSOPEN;

384         WR(rqp)->q_ptr = rqp->q_ptr = ptsp;

386         mutex_exit(&ptsp->pt_lock);
387         mutex_exit(&ptms_lock);
```

```
389         qprocson(rqp);

391         /*
392          * After qprocson pts driver is fully plumbed into the stream and can
393          * send/receive messages. Setting pts_rdq will allow master side to send
394          * messages to the slave. This setting can't occur before qprocson() is
395          * finished because slave is not ready to process them.
396          */
397         PT_ENTER_WRITE(ptsp);
398         ptsp->pts_rdq = rqp;
399         ASSERT(ptsp->pt_nullmsg == NULL);
400         ptsp->pt_nullmsg = mp;
401         PT_EXIT_WRITE(ptsp);

403         /*
404          * set up hi/lo water marks on stream head read queue
405          * and add controlling tty if not set
406          */

408         mop->b_datap->db_type = M_SETOPTS;
409         mop->b_wptr += sizeof (struct stroptions);
410         sop = (struct stroptions *)mop->b_rptr;
411         sop->so_flags = SO_HIWAT | SO_LOWAT | SO_ISTTY;
412         sop->so_hiwat = _TTY_BUFSIZ;
412         sop->so_hiwat = 512;
413         sop->so_lowat = 256;
414         putnext(rqp, mop);

416         return (0);
417 }
```

_____unchanged_portion_omitted_

```
**********************************************************
   28225 Thu Jul 27 16:35:47 2017
new/usr/src/uts/common/io/zcons.c
8527 tty buffer/queue sizes should be larger
**********************************************************
```

```
/*
 * 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 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */


/*
 * Zone Console Driver.
 *
 * This driver, derived from the pts/ptm drivers, is the pseudo console driver
 * for system zones.  Its implementation is straightforward.  Each instance
 * of the driver represents a global-zone/local-zone pair (this maps in a
 * straightforward way to the commonly used terminal notion of "master side"
 * and "slave side", and we use that terminology throughout).
 *
 * Instances of zcons are onlined as children of /pseudo/zconsnex@1/
 * by zoneadmd in userland, using the devctl framework; thus the driver
 * does not need to maintain any sort of "admin" node.
 *
 * The driver shuttles I/O from master side to slave side and back.  In a break
 * from the pts/ptm semantics, if one side is not open, I/O directed towards
 * it will simply be discarded.  This is so that if zoneadmd is not holding
 * the master side console open (i.e. it has died somehow), processes in
 * the zone do not experience any errors and I/O to the console does not
 * hang.
 *
 * TODO: we may want to revisit the other direction; i.e. we may want
 * zoneadmd to be able to detect whether no zone processes are holding the
 * console open, an unusual situation.
 *
 *
 *
 * MASTER SIDE IOCTLS
 *
 * The ZC_HOLDSLAVE and ZC_RELEASESLAVE ioctls instruct the master side of the
 * console to hold and release a reference to the slave side's vnode.  They are
 * meant to be issued by zoneadmd after the console device node is created and
 * before it is destroyed so that the slave's STREAMS anchor, ptem, is
 * preserved when ttymon starts popping STREAMS modules from within the
 * associated zone.  This guarantees that the zone console will always have
 * terminal semantics while the zone is running.
```



```
 *
 * Here is the issue: the ptem module is anchored in the zone console
 * (slave side) so that processes within the associated non-global zone will
 * fail to pop it off, thus ensuring that the slave will retain terminal
 * semantics.  When a process attempts to pop the anchor off of a stream, the
 * STREAMS subsystem checks whether the calling process' zone is the same as
 * that of the process that pushed the anchor onto the stream and cancels the
 * pop if they differ.  zoneadmd used to hold an open file descriptor for the
 * slave while the associated non-global zone ran, thus ensuring that the
 * slave's STREAMS anchor would never be popped from within the non-global zone
 * (because zoneadmd runs in the global zone).  However, this file descriptor
 * was removed to make zone console management more robust.  sad(7D) is now
 * used to automatically set up the slave's STREAMS modules when the zone
 * console is freshly opened within the associated non-global zone.  However,
 * when a process within the non-global zone freshly opens the zone console, the
 * anchor is pushed from within the non-global zone, making it possible for
 * processes within the non-global zone (e.g., ttymon) to pop the anchor and
 * destroy the zone console's terminal semantics.
 *
 * One solution is to make the zcons device hold the slave open while the
 * associated non-global zone runs so that the STREAMS anchor will always be
 * associated with the global zone.  Unfortunately, the slave cannot be opened
 * from within the zcons driver because the driver is not reentrant: it has
 * an outer STREAMS perimeter.  Therefore, the next best option is for zcons to
 * provide an ioctl interface to zoneadmd to manage holding and releasing
 * the slave side of the console.  It is sufficient to hold the slave side's
 * vnode and bump the associated snode's reference count to preserve the slave's
 * STREAMS configuration while the associated zone runs, so that's what the
 * ioctls do.
 *
 *
 * ZC_HOLDSLAVE
 *
 * This ioctl takes a file descriptor as an argument.  It effectively gets a
 * reference to the slave side's minor node's vnode and bumps the associated
 * snode's reference count.  The vnode reference is stored in the zcons device
 * node's soft state.  This ioctl succeeds if the given file descriptor refers
 * to the slave side's minor node or if there is already a reference to the
 * slave side's minor node's vnode in the device's soft state.
 *
 *
 * ZC_RELEASESLAVE
 *
 * This ioctl takes a file descriptor as an argument.  It effectively releases
 * the vnode reference stored in the zcons device node's soft state (which was
 * previously acquired via ZC_HOLDSLAVE) and decrements the reference count of
 * the snode associated with the vnode.  This ioctl succeeds if the given file
 * descriptor refers to the slave side's minor node or if no reference to the
 * slave side's minor node's vnode is stored in the device's soft state.
 *
 *
 * Note that the file descriptor arguments for both ioctls must be cast to
 * integers of pointer width.
 *
 * Here's how the dance between zcons and zoneadmd works:
 *
 *     Zone boot:
 *     1.  While booting the zone, zoneadmd creates an instance of zcons.
 *     2.  zoneadmd opens the master and slave sides of the new zone console
 *         and issues the ZC_HOLDSLAVE ioctl on the master side, passing its
 *         file descriptor for the slave side as the ioctl argument.
 *     3.  zcons holds the slave side's vnode, bumps the snode's reference
 *         count, and stores a pointer to the vnode in the device's soft
 *         state.
 *     4.  zoneadmd closes the master and slave sides and continues to boot
 *         the zone.
```

```
 128  *
 129  *     Zone halt:
 130  *     1.  While halting the zone, zoneadmd opens the master and slave sides
 131  *         of the zone's console and issues the ZC_RELEASESLAVE ioctl on the
 132  *         master side, passing its file descriptor for the slave side as the
 133  *         ioctl argument.
 134  *     2.  zcons decrements the slave side's snode's reference count, releases
 135  *         the slave's vnode, and eliminates its reference to the vnode in the
 136  *         device's soft state.
 137  *     3.  zoneadmd closes the master and slave sides.
 138  *     4.  zoneadmd destroys the zcons device and continues to halt the zone.
 139  *
 140  * It is necessary for zoneadmd to hold the slave open while issuing
 141  * ZC_RELEASESLAVE because zcons might otherwise release the last reference to
 142  * the slave's vnode.  If it does, then specfs will panic because it will expect
 143  * that the STREAMS configuration for the vnode was destroyed, which VN_RELE
 144  * doesn't do.  Forcing zoneadmd to hold the slave open guarantees that zcons
 145  * won't release the vnode's last reference.  zoneadmd will properly destroy the
 146  * vnode and the snode when it closes the file descriptor.
 147  *
 148  * Technically, any process that can access the master side can issue these
 149  * ioctls, but they should be treated as private interfaces for zoneadmd.
 150  */

 152 #include <sys/types.h>
 153 #include <sys/cmn_err.h>
 154 #include <sys/conf.h>
 155 #include <sys/cred.h>
 156 #include <sys/ddi.h>
 157 #include <sys/debug.h>
 158 #include <sys/devops.h>
 159 #include <sys/errno.h>
 160 #include <sys/file.h>
 161 #include <sys/kstr.h>
 162 #include <sys/modctl.h>
 163 #include <sys/param.h>
 164 #include <sys/stat.h>
 165 #include <sys/stream.h>
 166 #include <sys/stropts.h>
 167 #include <sys/strsun.h>
 168 #include <sys/sunddi.h>
 169 #include <sys/sysmacros.h>
 170 #include <sys/systm.h>
 171 #include <sys/types.h>
 172 #include <sys/zcons.h>
 173 #include <sys/vnode.h>
 174 #include <sys/fs/snode.h>
 175 #include <sys/zone.h>

 177 static int zc_getinfo(dev_info_t *, ddi_info_cmd_t, void *, void **);
 178 static int zc_attach(dev_info_t *, ddi_attach_cmd_t);
 179 static int zc_detach(dev_info_t *, ddi_detach_cmd_t);

 181 static int zc_open(queue_t *, dev_t *, int, int, cred_t *);
 182 static int zc_close(queue_t *, int, cred_t *);
 183 static void zc_wput(queue_t *, mblk_t *);
 184 static void zc_rsrv(queue_t *);
 185 static void zc_wsrv(queue_t *);

 187 /*
 188  * The instance number is encoded in the dev_t in the minor number; the lowest
 189  * bit of the minor number is used to track the master vs. slave side of the
 190  * virtual console.  The rest of the bits in the minor number are the instance.
 191  */
 192 #define ZC_MASTER_MINOR         0
 193 #define ZC_SLAVE_MINOR          1
```



```
 195 #define ZC_INSTANCE(x)          (getminor((x)) >> 1)
 196 #define ZC_NODE(x)              (getminor((x)) & 0x01)

 198 /*
 199  * This macro converts a zc_state_t pointer to the associated slave minor node's
 200  * dev_t.
 201  */
 202 #define ZC_STATE_TO_SLAVEDEV(x) (makedevice(ddi_driver_major((x)->zc_devinfo), \
 203         (minor_t)(ddi_get_instance((x)->zc_devinfo) << 1 | ZC_SLAVE_MINOR)))

 205 int zcons_debug = 0;
 206 #define DBG(a)   if (zcons_debug) cmn_err(CE_NOTE, a)
 207 #define DBG1(a, b)   if (zcons_debug) cmn_err(CE_NOTE, a, b)


 210 /*
 211  * Zone Console Pseudo Terminal Module: stream data structure definitions
 212  */
 213 static struct module_info zc_info = {
 214         31337,  /* c0z we r hAx0rs */
 215         "zcons",
 216         0,
 217         INFPSZ,
 218         _TTY_BUFSIZ,
 218         2048,
 219         128
 220 };
_____unchanged_portion_omitted_

 451 /*ARGSUSED*/
 452 static int
 453 zc_master_open(zc_state_t *zcs,
 454     queue_t     *rqp,   /* pointer to the read side queue */
 455     dev_t       *devp,  /* pointer to stream tail's dev */
 456     int         oflag,  /* the user open(2) supplied flags */
 457     int         sflag,  /* open state flag */
 458     cred_t      *credp) /* credentials */
 459 {
 460         mblk_t *mop;
 461         struct stroptions *sop;

 463         /*
 464          * Enforce exclusivity on the master side; the only consumer should
 465          * be the zoneadmd for the zone.
 466          */
 467         if ((zcs->zc_state & ZC_STATE_MOPEN) != 0)
 468                 return (EBUSY);

 470         if ((mop = allocb(sizeof (struct stroptions), BPRI_MED)) == NULL) {
 471                 DBG("zc_master_open(): mop allocation failed\n");
 472                 return (ENOMEM);
 473         }

 475         zcs->zc_state |= ZC_STATE_MOPEN;

 477         /*
 478          * q_ptr stores driver private data; stash the soft state data on both
 479          * read and write sides of the queue.
 480          */
 481         WR(rqp)->q_ptr = rqp->q_ptr = zcs;
 482         qprocson(rqp);

 484         /*
 485          * Following qprocson(), the master side is fully plumbed into the
 486          * STREAM and may send/receive messages.  Setting zcs->zc_master_rdq
```

```
487          * will allow the slave to send messages to us (the master).
488          * This cannot occur before qprocson() because the master is not
489          * ready to process them until that point.
490          */
491         zcs->zc_master_rdq = rqp;

493         /*
494          * set up hi/lo water marks on stream head read queue and add
495          * controlling tty as needed.
496          */
497         mop->b_datap->db_type = M_SETOPTS;
498         mop->b_wptr += sizeof (struct stroptions);
499         sop = (struct stroptions *)(void *)mop->b_rptr;
500         if (oflag & FNOCTTY)
501                 sop->so_flags = SO_HIWAT | SO_LOWAT;
502         else
503                 sop->so_flags = SO_HIWAT | SO_LOWAT | SO_ISTTY;
504         sop->so_hiwat = _TTY_BUFSIZ;
504         sop->so_hiwat = 512;
505         sop->so_lowat = 256;
506         putnext(rqp, mop);

508         return (0);
509 }

511 /*ARGSUSED*/
512 static int
513 zc_slave_open(zc_state_t *zcs,
514     queue_t     *rqp,   /* pointer to the read side queue */
515     dev_t       *devp,  /* pointer to stream tail's dev */
516     int         oflag,  /* the user open(2) supplied flags */
517     int         sflag,  /* open state flag */
518     cred_t      *credp) /* credentials */
519 {
520         mblk_t *mop;
521         struct stroptions *sop;
522         major_t major;
523         minor_t minor;
524         minor_t lastminor;
525         uint_t anchorindex;

527         /*
528          * The slave side can be opened as many times as needed.
529          */
530         if ((zcs->zc_state & ZC_STATE_SOPEN) != 0) {
531                 ASSERT((rqp != NULL) && (WR(rqp)->q_ptr == zcs));
532                 return (0);
533         }

535         /*
536          * Set up sad(7D) so that the necessary STREAMS modules will be in
537          * place.  A wrinkle is that 'ptem' must be anchored
538          * in place (see streamio(7i)) because we always want the console to
539          * have terminal semantics.
540          */
541         minor = ddi_get_instance(zcs->zc_devinfo) << 1 | ZC_SLAVE_MINOR;
542         major = ddi_driver_major(zcs->zc_devinfo);
543         lastminor = 0;
544         anchorindex = 1;
545         if (kstr_autopush(SET_AUTOPUSH, &major, &minor, &lastminor,
546             &anchorindex, zcons_mods) != 0) {
547                 DBG("zc_slave_open(): kstr_autopush() failed\n");
548                 return (EIO);
549         }

551         if ((mop = allocb(sizeof (struct stroptions), BPRI_MED)) == NULL) {
```



```
552                 DBG("zc_slave_open(): mop allocation failed\n");
553                 return (ENOMEM);
554         }

556         zcs->zc_state |= ZC_STATE_SOPEN;

558         /*
559          * q_ptr stores driver private data; stash the soft state data on both
560          * read and write sides of the queue.
561          */
562         WR(rqp)->q_ptr = rqp->q_ptr = zcs;

564         qprocson(rqp);

566         /*
567          * Must follow qprocson(), since we aren't ready to process until then.
568          */
569         zcs->zc_slave_rdq = rqp;

571         /*
572          * set up hi/lo water marks on stream head read queue and add
573          * controlling tty as needed.
574          */
575         mop->b_datap->db_type = M_SETOPTS;
576         mop->b_wptr += sizeof (struct stroptions);
577         sop = (struct stroptions *)(void *)mop->b_rptr;
578         sop->so_flags = SO_HIWAT | SO_LOWAT | SO_ISTTY;
579         sop->so_hiwat = _TTY_BUFSIZ;
579         sop->so_hiwat = 512;
580         sop->so_lowat = 256;
581         putnext(rqp, mop);

583         return (0);
584 }
_____unchanged_portion_omitted_
```

```
**********************************************************
   14512 Thu Jul 27 16:35:48 2017
new/usr/src/uts/common/sys/param.h
8527 tty buffer/queue sizes should be larger
**********************************************************
```

```
/*
 * 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 2014 Nexenta Systems, Inc.  All rights reserved.
 * Copyright (c) 1998, 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*      Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T     */
/*        All Rights Reserved   */

/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

#ifndef _SYS_PARAM_H
#define _SYS_PARAM_H

#ifndef _ASM            /* Avoid typedef headaches for assembly files */
#include <sys/types.h>
#include <sys/isa_defs.h>
#endif /* _ASM */

#include <sys/null.h>

#ifdef __cplusplus
extern "C" {
#endif

/*
 * Fundamental variables; don't change too often.
 */

/*
 * _POSIX_VDISABLE has historically been defined in <sys/param.h> since
 * an early merge with AT&T source.  It has also historically been defined
 * in <sys/termios.h>. The POSIX standard, IEEE Std. 1003.1-1988 initially
```



```
 * required the existence of _POSIX_VDISABLE in <sys/termios.h>.
 * Subsequent versions of the IEEE Standard as well as the X/Open
 * specifications required that _POSIX_VDISABLE be defined in <unistd.h>
 * while still allowing for it's existence in other headers.  With the
 * introduction of XPG6, _POSIX_VDISABLE can only be defined in <unistd.h>.
 */
#if !defined(_XPG6) || defined(__EXTENSIONS__)
#ifndef _POSIX_VDISABLE
#define _POSIX_VDISABLE 0       /* Disable special character functions */
#endif
#endif /* !defined(_XPG6) || defined(__EXTENSIONS__) */

/* The actual size of the TTY input queue */
#define _TTY_BUFSIZ     2048

/*
 * These defines all have their historical value.  The actual size of the tty
 * buffer both for the line-editor in ldterm, and in general, is above as
 * _TTY_BUFSIZ.
 *
 * We leave these defines at their historical value to match the behaviour of
 * BSD and Linux.
 */
#endif /* ! codereview */
#ifndef MAX_INPUT
#define MAX_INPUT       512     /* Maximum bytes stored in the input queue */
#endif

#ifndef MAX_CANON
#define MAX_CANON       256     /* Maximum bytes for canonical processing */
#endif
#define CANBSIZ         256     /* max size of typewriter line  */

#endif /* ! codereview */

#define UID_NOBODY      60001   /* user ID no body */
#define GID_NOBODY      UID_NOBODY
#define UID_UNKNOWN     96
#define GID_UNKNOWN     UID_UNKNOWN
#define UID_DLADM       15
#define UID_NETADM      16
#define GID_NETADM      65
#define UID_NOACCESS    60002   /* user ID no access */

#ifdef _KERNEL
#define MAX_TASKID      999999
#define MAX_MAXPID      999999
#define MAXEPHUID       0xfffffffcu     /* max ephemeral user id */

#define FAMOUS_PID_SCHED        0
#define FAMOUS_PID_INIT         1
#define FAMOUS_PID_PAGEOUT      2
#define FAMOUS_PID_FSFLUSH      3
#define FAMOUS_PIDS             4
#endif

#ifdef DEBUG
#define DEFAULT_MAXPID  999999
#define DEFAULT_JUMPPID 100000
#else
#define DEFAULT_MAXPID  30000
#define DEFAULT_JUMPPID 0
#endif

#define MAXUID          2147483647      /* max user id */
```

```
127 #define MAXPROJID       MAXUID          /* max project id */
128 #define MAXLINK         32767   /* max links */

130 #define MINEPHUID       0x80000000u     /* min ephemeral user id */

132 #define NMOUNT          40      /* est. of # mountable fs for quota calc */

 78 #define CANBSIZ         256     /* max size of typewriter line  */

134 #define NOFILE          20      /* this define is here for      */
135                                 /* compatibility purposes only  */
136                                 /* and will be removed in a     */
137                                 /* later release                */

139 /*
140  * These define the maximum and minimum allowable values of the
141  * configurable parameter NGROUPS_MAX.
142  */
143 #define NGROUPS_UMIN    0
144 #define NGROUPS_UMAX    1024
145 #define NGROUPS_OLDMAX  32

147 /*
148  * NGROUPS_MAX_DEFAULT: *MUST* match NGROUPS_MAX value in limits.h.
149  */
150 #define NGROUPS_MAX_DEFAULT     16

152 /*
153  * Default process priority.  Keep it in sync with limits.h.
154  */
155 #define NZERO   20

157 /*
158  * Fundamental constants of the implementation--cannot be changed easily.
159  */

161 #define NBPW    sizeof (int)    /* number of bytes in an integer */

163 #define CMASK   022             /* default mask for file creation */
164 #define CDLIMIT (1L<<11)        /* default max write address */
165 #define NBPS            0x20000 /* Number of bytes per segment */
166 #define NBPSCTR         512     /* Bytes per disk sector.       */
167 #define UBSIZE          512     /* unix block size.             */
168 #define SCTRSHFT        9       /* Shift for BPSECT.            */

170 #ifdef _LITTLE_ENDIAN
171 #define lobyte(X)       (((unsigned char *)&(X))[0])
172 #define hibyte(X)       (((unsigned char *)&(X))[1])
173 #define loword(X)       (((ushort_t *)&(X))[0])
174 #define hiword(X)       (((ushort_t *)&(X))[1])
175 #endif
176 #ifdef _BIG_ENDIAN
177 #define lobyte(X)       (((unsigned char *)&(X))[1])
178 #define hibyte(X)       (((unsigned char *)&(X))[0])
179 #define loword(X)       (((ushort_t *)&(X))[1])
180 #define hiword(X)       (((ushort_t *)&(X))[0])
181 #endif

183 /* REMOTE -- whether machine is primary, secondary, or regular */
184 #define SYSNAME 9               /* # chars in system name */
185 #define PREMOTE 39

187 /*
188  * MAXPATHLEN defines the longest permissible path length,
189  * including the terminating null, after expanding symbolic links.
190  * TYPICALMAXPATHLEN is used in a few places as an optimization
```



```
191  * with a local buffer on the stack to avoid kmem_alloc().
192  * MAXSYMLINKS defines the maximum number of symbolic links
193  * that may be expanded in a path name. It should be set high
194  * enough to allow all legitimate uses, but halt infinite loops
195  * reasonably quickly.
196  * MAXNAMELEN is the length (including the terminating null) of
197  * the longest permissible file (component) name.
198  */
199 #define MAXPATHLEN      1024
200 #define TYPICALMAXPATHLEN       64
201 #define MAXSYMLINKS     20
202 #define MAXNAMELEN      256

204 /*
205  * MAXLINKNAMELEN defines the longest possible permitted datalink name,
206  * including the terminating NUL.  Note that this must not be larger
207  * than related networking constants such as LIFNAMSIZ.
208  */
209 #define MAXLINKNAMELEN  32

211 #ifndef NADDR
212 #define NADDR 13
213 #endif

215 /*
216  * The following are defined to be the same as
217  * defined in /usr/include/limits.h.  They are
218  * needed for pipe and FIFO compatibility.
219  */
220 #ifndef PIPE_BUF        /* max # bytes atomic in write to a pipe */
221 #define PIPE_BUF        5120
222 #endif  /* PIPE_BUF */

224 #ifndef PIPE_MAX        /* max # bytes written to a pipe in a write */
225 #define PIPE_MAX        5120
226 #endif  /* PIPE_MAX */

228 #ifndef NBBY
229 #define NBBY    8                       /* number of bits per byte */
230 #endif

232 /* macros replacing interleaving functions */
233 #define dkblock(bp)     ((bp)->b_blkno)
234 #define dkunit(bp)      (minor((bp)->b_dev) >> 3)

236 /*
237  * File system parameters and macros.
238  *
239  * The file system is made out of blocks of at most MAXBSIZE units,
240  * with smaller units (fragments) only in the last direct block.
241  * MAXBSIZE primarily determines the size of buffers in the buffer
242  * pool. It may be made larger without any effect on existing
243  * file systems; however making it smaller make make some file
244  * systems unmountable.
245  *
246  * Note that the blocked devices are assumed to have DEV_BSIZE
247  * "sectors" and that fragments must be some multiple of this size.
248  */
249 #define MAXBSIZE        8192
250 #define DEV_BSIZE       512
251 #define DEV_BSHIFT      9               /* log2(DEV_BSIZE) */
252 #define MAXFRAG         8
253 #ifdef  _SYSCALL32
254 #define MAXOFF32_T      0x7fffffff
255 #endif
256 #ifdef  _LP64
```

```
257 #define MAXOFF_T        0x7fffffffffffffffl
258 #define MAXOFFSET_T     0x7fffffffffffffffl
259 #else
260 #define MAXOFF_T        0x7fffffffl
261 #ifdef _LONGLONG_TYPE
262 #define MAXOFFSET_T     0x7fffffffffffffffLL
263 #else
264 #define MAXOFFSET_T     0x7fffffff
265 #endif
266 #endif  /* _LP64 */

268 #define btodb(bytes)                    /* calculates (bytes / DEV_BSIZE) */ \
269         ((unsigned long)(bytes) >> DEV_BSHIFT)
270 #define dbtob(db)                       /* calculates (db * DEV_BSIZE) */ \
271         ((unsigned long)(db) << DEV_BSHIFT)

273 /*      64 bit versions of btodb and dbtob */
274 #define lbtodb(bytes)                   /* calculates (bytes / DEV_BSIZE) */ \
275         ((u_offset_t)(bytes) >> DEV_BSHIFT)
276 #define ldbtob(db)                      /* calculates (db * DEV_BSIZE) */ \
277         ((u_offset_t)(db) << DEV_BSHIFT)

279 #ifndef _ASM    /* Avoid typedef headaches for assembly files */
280 #ifndef NODEV
281 #define NODEV   (dev_t)(-1l)
282 #ifdef _SYSCALL32
283 #define NODEV32 (dev32_t)(-1)
284 #endif  /* _SYSCALL32 */
285 #endif  /* NODEV */
286 #endif  /* _ASM */

288 /*
289  * Size of arg list passed in by user.
290  */
291 #define NCARGS32        0x100000
292 #define NCARGS64        0x200000
293 #ifdef  _LP64
294 #define NCARGS          NCARGS64
295 #else   /* _LP64 */
296 #define NCARGS          NCARGS32
297 #endif  /* _LP64 */

299 /*
300  * Scale factor for scaled integers used to count
301  * %cpu time and load averages.
302  */
303 #define FSHIFT  8               /* bits to right of fixed binary point */
304 #define FSCALE  (1<<FSHIFT)

306 /*
307  * Delay units are in microseconds.
308  *
309  * XXX  These macros are not part of the DDI!
310  */
311 #if defined(_KERNEL) && !defined(_ASM)
312 extern void drv_usecwait(clock_t);
313 #define DELAY(n)        drv_usecwait(n)
314 #define CDELAY(c, n)    \
315 { \
316         register int N = n; \
317         while (--N > 0) { \
318                 if (c) \
319                         break; \
320                 drv_usecwait(1); \
321         } \
322 }
```

_____***unchanged_portion_omitted_***