```
**********************************************************
  144451 Mon Jan 14 13:17:32 2019
new/usr/src/uts/common/io/ldterm.c
10088 ldterm_do_ioctl() shouldn't check for a NULL array
**********************************************************
```

```
/*
 * 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) 2018, Joyent, Inc.
 * Copyright (c) 2014, Joyent, Inc.  All rights reserved.
 * Copyright 2018 OmniOS Community Edition (OmniOSce) Association.
 */

/* Copyright (c) 1984, 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.
 */

/*
 * Standard Streams Terminal Line Discipline module.
 */

#include <sys/param.h>
#include <sys/types.h>
#include <sys/termio.h>
#include <sys/stream.h>
#include <sys/conf.h>
#include <sys/stropts.h>
#include <sys/strsubr.h>
#include <sys/strsun.h>
#include <sys/strtty.h>
#include <sys/signal.h>
#include <sys/file.h>
#include <sys/errno.h>
#include <sys/debug.h>
#include <sys/cmn_err.h>
#include <sys/euc.h>
#include <sys/eucioctl.h>
```



```
#include <sys/csiioctl.h>
#include <sys/ptms.h>
#include <sys/ldterm.h>
#include <sys/cred.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/kmem.h>
#include <sys/modctl.h>

/* Time limit when draining during a close(9E) invoked by exit(2) */
/* Can be set to zero to emulate the old, broken behavior */
int ldterm_drain_limit = 15000000;

/*
 * Character types.
 */
#define ORDINARY        0
#define CONTROL         1
#define BACKSPACE       2
#define NEWLINE         3
#define TAB             4
#define VTAB            5
#define RETURN          6

/*
 * The following for EUC handling:
 */
#define T_SS2           7
#define T_SS3           8

/*
 * Table indicating character classes to tty driver.  In particular,
 * if the class is ORDINARY, then the character needs no special
 * processing on output.
 *
 * Characters in the C1 set are all considered CONTROL; this will
 * work with terminals that properly use the ANSI/ISO extensions,
 * but might cause distress with terminals that put graphics in
 * the range 0200-0237.  On the other hand, characters in that
 * range cause even greater distress to other UNIX terminal drivers....
 */

static char typetab[256] = {
/* 000 */       CONTROL,        CONTROL,        CONTROL,        CONTROL,
/* 004 */       CONTROL,        CONTROL,        CONTROL,        CONTROL,
/* 010 */       BACKSPACE,      TAB,            NEWLINE,        CONTROL,
/* 014 */       VTAB,           RETURN,         CONTROL,        CONTROL,
/* 020 */       CONTROL,        CONTROL,        CONTROL,        CONTROL,
/* 024 */       CONTROL,        CONTROL,        CONTROL,        CONTROL,
/* 030 */       CONTROL,        CONTROL,        CONTROL,        CONTROL,
/* 034 */       CONTROL,        CONTROL,        CONTROL,        CONTROL,
/* 040 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
/* 044 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
/* 050 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
/* 054 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
/* 060 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
/* 064 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
/* 070 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
/* 074 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
/* 100 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
/* 104 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
/* 110 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
/* 114 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
/* 120 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
/* 124 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
/* 130 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
```

```
 127 /* 134 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 128 /* 140 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 129 /* 144 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 130 /* 150 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 131 /* 154 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 132 /* 160 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 133 /* 164 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 134 /* 170 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 135 /* 174 */       ORDINARY,       ORDINARY,       ORDINARY,       CONTROL,
 136 /* 200 */       CONTROL,        CONTROL,        CONTROL,        CONTROL,
 137 /* 204 */       CONTROL,        CONTROL,        T_SS2,          T_SS3,
 138 /* 210 */       CONTROL,        CONTROL,        CONTROL,        CONTROL,
 139 /* 214 */       CONTROL,        CONTROL,        CONTROL,        CONTROL,
 140 /* 220 */       CONTROL,        CONTROL,        CONTROL,        CONTROL,
 141 /* 224 */       CONTROL,        CONTROL,        CONTROL,        CONTROL,
 142 /* 230 */       CONTROL,        CONTROL,        CONTROL,        CONTROL,
 143 /* 234 */       CONTROL,        CONTROL,        CONTROL,        CONTROL,
 144 /* 240 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 145 /* 244 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 146 /* 250 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 147 /* 254 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 148 /* 260 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 149 /* 264 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 150 /* 270 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 151 /* 274 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 152 /* 300 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 153 /* 304 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 154 /* 310 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 155 /* 314 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 156 /* 320 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 157 /* 324 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 158 /* 330 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 159 /* 334 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 160 /* 340 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 161 /* 344 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 162 /* 350 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 163 /* 354 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 164 /* 360 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 165 /* 364 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 166 /* 370 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 167 /*
 168  * WARNING:  For EUC, 0xFF must be an ordinary character.  It is used with
 169  * single-byte EUC in some of the "ISO Latin Alphabet" codesets, and occupies
 170  * a screen position; in those ISO sets where that position isn't used, it
 171  * shouldn't make any difference.
 172  */
 173 /* 374 */       ORDINARY,       ORDINARY,       ORDINARY,       ORDINARY,
 174 };
```

*unchanged_portion_omitted_*

```
4096 /*
4097  * Called when an M_IOCTL message is seen on the write queue; does
4098  * whatever we're supposed to do with it, and either replies
4099  * immediately or passes it to the next module down.
4100  */
4101 static void
4102 ldterm_do_ioctl(queue_t *q, mblk_t *mp)
4103 {
4104         ldtermstd_state_t *tp;
4105         struct iocblk *iocp;
4106         struct eucioc *euciocp; /* needed for EUC ioctls */
4107         ldterm_cs_data_user_t *csdp;
4108         int i;
4109         int locale_name_sz;
4110         uchar_t maxbytelen;
```



```
4111         uchar_t maxscreenlen;
4112         int error;

4114         iocp = (struct iocblk *)mp->b_rptr;
4115         tp = (ldtermstd_state_t *)q->q_ptr;

4117         switch (iocp->ioc_cmd) {

4119         case TCSETS:
4120         case TCSETSW:
4121         case TCSETSF:
4122                 {
4123                         /*
4124                          * Set current parameters and special
4125                          * characters.
4126                          */
4127                         struct termios *cb;
4128                         struct termios oldmodes;

4130                         error = miocpullup(mp, sizeof (struct termios));
4131                         if (error != 0) {
4132                                 miocnak(q, mp, 0, error);
4133                                 return;
4134                         }

4136                         cb = (struct termios *)mp->b_cont->b_rptr;

4138                         oldmodes = tp->t_amodes;
4139                         tp->t_amodes = *cb;
4140                         if ((tp->t_amodes.c_lflag & PENDIN) &&
4141                             (tp->t_modes.c_lflag & IEXTEN)) {
4142                                 /*
4143                                  * Yuk.  The C shell file completion
4144                                  * code actually uses this "feature",
4145                                  * so we have to support it.
4146                                  */
4147                                 if (tp->t_message != NULL) {
4148                                         tp->t_state |= TS_RESCAN;
4149                                         qenable(RD(q));
4150                                 }
4151                                 tp->t_amodes.c_lflag &= ~PENDIN;
4152                         }
4153                         bcopy(tp->t_amodes.c_cc, tp->t_modes.c_cc, NCCS);

4155                         /*
4156                          * ldterm_adjust_modes does not deal with
4157                          * cflags
4158                          */
4159                         tp->t_modes.c_cflag = tp->t_amodes.c_cflag;

4161                         ldterm_adjust_modes(tp);
4162                         if (chgstropts(&oldmodes, tp, RD(q)) == (-1)) {
4163                                 miocnak(q, mp, 0, EAGAIN);
4164                                 return;
4165                         }
4166                         /*
4167                          * The driver may want to know about the
4168                          * following iflags: IGNBRK, BRKINT, IGNPAR,
4169                          * PARMRK, INPCK, IXON, IXANY.
4170                          */
4171                         break;
4172                 }

4174         case TCSETA:
4175         case TCSETAW:
4176         case TCSETAF:
```

```
4177                {
4178                        /*
4179                         * Old-style "ioctl" to set current
4180                         * parameters and special characters. Don't
4181                         * clear out the unset portions, leave them
4182                         * as they are.
4183                         */
4184                        struct termio *cb;
4185                        struct termios oldmodes;

4187                        error = miocpullup(mp, sizeof (struct termio));
4188                        if (error != 0) {
4189                                miocnak(q, mp, 0, error);
4190                                return;
4191                        }

4193                        cb = (struct termio *)mp->b_cont->b_rptr;

4195                        oldmodes = tp->t_amodes;
4196                        tp->t_amodes.c_iflag =
4197                            (tp->t_amodes.c_iflag & 0xffff0000 | cb->c_iflag);
4198                        tp->t_amodes.c_oflag =
4199                            (tp->t_amodes.c_oflag & 0xffff0000 | cb->c_oflag);
4200                        tp->t_amodes.c_cflag =
4201                            (tp->t_amodes.c_cflag & 0xffff0000 | cb->c_cflag);
4202                        tp->t_amodes.c_lflag =
4203                            (tp->t_amodes.c_lflag & 0xffff0000 | cb->c_lflag);

4205                        bcopy(cb->c_cc, tp->t_modes.c_cc, NCC);
4206                        /* TCGETS returns amodes, so update that too */
4207                        bcopy(cb->c_cc, tp->t_amodes.c_cc, NCC);

4209                        /* ldterm_adjust_modes does not deal with cflags */

4211                        tp->t_modes.c_cflag = tp->t_amodes.c_cflag;

4213                        ldterm_adjust_modes(tp);
4214                        if (chgstropts(&oldmodes, tp, RD(q)) == (-1)) {
4215                                miocnak(q, mp, 0, EAGAIN);
4216                                return;
4217                        }
4218                        /*
4219                         * The driver may want to know about the
4220                         * following iflags: IGNBRK, BRKINT, IGNPAR,
4221                         * PARMRK, INPCK, IXON, IXANY.
4222                         */
4223                        break;
4224                }

4226        case TCFLSH:
4227                /*
4228                 * Do the flush on the write queue immediately, and
4229                 * queue up any flush on the read queue for the
4230                 * service procedure to see.  Then turn it into the
4231                 * appropriate M_FLUSH message, so that the module
4232                 * below us doesn't have to know about TCFLSH.
4233                 */
4234                error = miocpullup(mp, sizeof (int));
4235                if (error != 0) {
4236                        miocnak(q, mp, 0, error);
4237                        return;
4238                }

4240                ASSERT(mp->b_datap != NULL);
4241                if (*(int *)mp->b_cont->b_rptr == 0) {
4242                        ASSERT(mp->b_datap != NULL);
```



```
4243                        (void) putnextctl1(q, M_FLUSH, FLUSHR);
4244                        (void) putctl1(RD(q), M_FLUSH, FLUSHR);
4245                } else if (*(int *)mp->b_cont->b_rptr == 1) {
4246                        flushq(q, FLUSHDATA);
4247                        ASSERT(mp->b_datap != NULL);
4248                        tp->t_state |= TS_FLUSHWAIT;
4249                        (void) putnextctl1(RD(q), M_FLUSH, FLUSHW);
4250                        (void) putnextctl1(q, M_FLUSH, FLUSHW);
4251                } else if (*(int *)mp->b_cont->b_rptr == 2) {
4252                        flushq(q, FLUSHDATA);
4253                        ASSERT(mp->b_datap != NULL);
4254                        (void) putnextctl1(q, M_FLUSH, FLUSHRW);
4255                        tp->t_state |= TS_FLUSHWAIT;
4256                        (void) putnextctl1(RD(q), M_FLUSH, FLUSHRW);
4257                } else {
4258                        miocnak(q, mp, 0, EINVAL);
4259                        return;
4260                }
4261                ASSERT(mp->b_datap != NULL);
4262                iocp->ioc_rval = 0;
4263                miocack(q, mp, 0, 0);
4264                return;

4266        case TCXONC:
4267                error = miocpullup(mp, sizeof (int));
4268                if (error != 0) {
4269                        miocnak(q, mp, 0, error);
4270                        return;
4271                }

4273                switch (*(int *)mp->b_cont->b_rptr) {
4274                case 0:
4275                        if (!(tp->t_state & TS_TTSTOP)) {
4276                                (void) putnextctl(q, M_STOP);
4277                                tp->t_state |= (TS_TTSTOP|TS_OFBLOCK);
4278                        }
4279                        break;

4281                case 1:
4282                        if (tp->t_state & TS_TTSTOP) {
4283                                (void) putnextctl(q, M_START);
4284                                tp->t_state &= ~(TS_TTSTOP|TS_OFBLOCK);
4285                        }
4286                        break;

4288                case 2:
4289                        (void) putnextctl(q, M_STOPI);
4290                        tp->t_state |= (TS_TBLOCK|TS_IFBLOCK);
4291                        break;

4293                case 3:
4294                        (void) putnextctl(q, M_STARTI);
4295                        tp->t_state &= ~(TS_TBLOCK|TS_IFBLOCK);
4296                        break;

4298                default:
4299                        miocnak(q, mp, 0, EINVAL);
4300                        return;
4301                }
4302                ASSERT(mp->b_datap != NULL);
4303                iocp->ioc_rval = 0;
4304                miocack(q, mp, 0, 0);
4305                return;
4306                /*
4307                 * TCSBRK is expected to be handled by the driver.
4308                 * The reason its left for the driver is that when
```

```
4309                  * the argument to TCSBRK is zero driver has to drain
4310                  * the data and sending a M_IOCACK from LDTERM before
4311                  * the driver drains the data is going to cause
4312                  * problems.
4313                  */
4315                 /*
4316                  * The following are EUC related ioctls.  For
4317                  * EUC_WSET, we have to pass the information on, even
4318                  * though we ACK the call.  It's vital in the EUC
4319                  * environment that everybody downstream knows about
4320                  * the EUC codeset widths currently in use; we
4321                  * therefore pass down the information in an M_CTL
4322                  * message.  It will bottom out in the driver.
4323                  */
4324         case EUC_WSET:
4325                 {
4327                         /* only needed for EUC_WSET */
4328                         struct iocblk *riocp;
4330                         mblk_t *dmp, *dmp_cont;
4332                         /*
4333                          * If the user didn't supply any information,
4334                          * NAK it.
4335                          */
4336                         error = miocpullup(mp, sizeof (struct eucioc));
4337                         if (error != 0) {
4338                                 miocnak(q, mp, 0, error);
4339                                 return;
4340                         }
4342                         euciocp = (struct eucioc *)mp->b_cont->b_rptr;
4343                         /*
4344                          * Check here for something reasonable.  If
4345                          * anything will take more than EUC_MAXW
4346                          * columns or more than EUC_MAXW bytes
4347                          * following SS2 or SS3, then just reject it
4348                          * out of hand. It's not impossible for us to
4349                          * do it, it just isn't reasonable.  So far,
4350                          * in the world, we've seen the absolute max
4351                          * columns to be 2 and the max number of
4352                          * bytes to be 3.  This allows room for some
4353                          * expansion of that, but it probably won't
4354                          * even be necessary. At the moment, we
4355                          * return a "range" error.  If you really
4356                          * need to, you can push EUC_MAXW up to over
4357                          * 200; it doesn't make sense, though, with
4358                          * only a CANBSIZ sized input limit (usually
4359                          * 256)!
4360                          */
4361                         for (i = 0; i < 4; i++) {
4362                                 if ((euciocp->eucw[i] > EUC_MAXW) ||
4363                                     (euciocp->scrw[i] > EUC_MAXW)) {
4364                                         miocnak(q, mp, 0, ERANGE);
4365                                         return;
4366                                 }
4367                         }
4368                         /*
4369                          * Otherwise, save the information in tp,
4370                          * force codeset 0 (ASCII) to be one byte,
4371                          * one column.
4372                          */
4373                         cp_eucwioc(euciocp, &tp->eucwioc, EUCIN);
4374                         tp->eucwioc.eucw[0] = tp->eucwioc.scrw[0] = 1;
```



```
4375                         /*
4376                          * Now, check out whether we're doing
4377                          * multibyte processing. if we are, we need
4378                          * to allocate a block to hold the parallel
4379                          * array. By convention, we've been passed
4380                          * what amounts to a CSWIDTH definition.  We
4381                          * actually NEED the number of bytes for
4382                          * Codesets 2 & 3.
4383                          */
4384                         tp->t_maxeuc = 0;       /* reset to say we're NOT */
4386                         tp->t_state &= ~TS_MEUC;
4387                         /*
4388                          * We'll set TS_MEUC if we're doing
4389                          * multi-column OR multi- byte OR both.  It
4390                          * makes things easier...  NOTE:  If we fail
4391                          * to get the buffer we need to hold display
4392                          * widths, then DON'T let the TS_MEUC bit get
4393                          * set!
4394                          */
4395                         for (i = 0; i < 4; i++) {
4396                                 if (tp->eucwioc.eucw[i] > tp->t_maxeuc)
4397                                         tp->t_maxeuc = tp->eucwioc.eucw[i];
4398                                 if (tp->eucwioc.scrw[i] > 1)
4399                                         tp->t_state |= TS_MEUC;
4400                         }
4401                         if ((tp->t_maxeuc > 1) || (tp->t_state & TS_MEUC)) {
4402                                 if (!tp->t_eucp_mp) {
4403                                         if ((tp->t_eucp_mp = allocb(_TTY_BUFSIZ,
4404                                             BPRI_HI)) == NULL) {
4405                                                 tp->t_maxeuc = 1;
4406                                                 tp->t_state &= ~TS_MEUC;
4407                                                 cmn_err(CE_WARN,
4408                                                     "Can't allocate eucp_mp");
4409                                                 miocnak(q, mp, 0, ENOSR);
4410                                                 return;
4411                                         }
4412                                         /*
4413                                          * here, if there's junk in
4414                                          * the canonical buffer, then
4415                                          * move the eucp pointer past
4416                                          * it, so we don't run off
4417                                          * the beginning.  This is a
4418                                          * total botch, but will
4419                                          * hopefully keep stuff from
4420                                          * getting too messed up
4421                                          * until the user flushes
4422                                          * this line!
4423                                          */
4424                                         if (tp->t_msglen) {
4425                                                 tp->t_eucp =
4426                                                     tp->t_eucp_mp->b_rptr;
4427                                                 for (i = tp->t_msglen; i; i--)
4428                                                         *tp->t_eucp++ = 1;
4429                                         } else {
4430                                                 tp->t_eucp =
4431                                                     tp->t_eucp_mp->b_rptr;
4432                                         }
4433                                 }
4434                                 /* doing multi-byte handling */
4435                                 tp->t_state |= TS_MEUC;
4437                         } else if (tp->t_eucp_mp) {
4438                                 freemsg(tp->t_eucp_mp);
4439                                 tp->t_eucp_mp = NULL;
4440                                 tp->t_eucp = NULL;
```

```
4441                    }
4442
4443                    /*
4444                     * Save the EUC width data we have at
4445                     * the t_csdata, set t_csdata.codeset_type to
4446                     * EUC one, and, switch the codeset methods at
4447                     * t_csmethods.
4448                     */
4449                    bzero(&tp->t_csdata.eucpc_data,
4450                        (sizeof (ldterm_eucpc_data_t) *
4451                        LDTERM_CS_MAX_CODESETS));
4452                    tp->t_csdata.eucpc_data[0].byte_length =
4453                        tp->eucwioc.eucw[1];
4454                    tp->t_csdata.eucpc_data[0].screen_width =
4455                        tp->eucwioc.scrw[1];
4456                    tp->t_csdata.eucpc_data[1].byte_length =
4457                        tp->eucwioc.eucw[2];
4458                    tp->t_csdata.eucpc_data[1].screen_width =
4459                        tp->eucwioc.scrw[2];
4460                    tp->t_csdata.eucpc_data[2].byte_length =
4461                        tp->eucwioc.eucw[3];
4462                    tp->t_csdata.eucpc_data[2].screen_width =
4463                        tp->eucwioc.scrw[3];
4464                    tp->t_csdata.version = LDTERM_DATA_VERSION;
4465                    tp->t_csdata.codeset_type = LDTERM_CS_TYPE_EUC;
4466                    /*
4467                     * We are not using the 'csinfo_num' anyway if the
4468                     * current codeset type is EUC. So, set it to
4469                     * the maximum possible.
4470                     */
4471                    tp->t_csdata.csinfo_num =
4472                        LDTERM_CS_TYPE_EUC_MAX_SUBCS;
4473                    if (tp->t_csdata.locale_name != (char *)NULL) {
4474                            kmem_free(tp->t_csdata.locale_name,
4475                                strlen(tp->t_csdata.locale_name) + 1);
4476                            tp->t_csdata.locale_name = (char *)NULL;
4477                    }
4478                    tp->t_csmethods = cs_methods[LDTERM_CS_TYPE_EUC];
4479
4480                    /*
4481                     * If we are able to allocate two blocks (the
4482                     * iocblk and the associated data), then pass
4483                     * it downstream, otherwise we'll need to NAK
4484                     * it, and drop whatever we WERE able to
4485                     * allocate.
4486                     */
4487                    if ((dmp = mkiocb(EUC_WSET)) == NULL) {
4488                            miocnak(q, mp, 0, ENOSR);
4489                            return;
4490                    }
4491                    if ((dmp_cont = allocb(EUCSIZE, BPRI_HI)) == NULL) {
4492                            freemsg(dmp);
4493                            miocnak(q, mp, 0, ENOSR);
4494                            return;
4495                    }
4496
4497                    /*
4498                     * We got both buffers.  Copy out the EUC
4499                     * information (as we received it, not what
4500                     * we're using!) & pass it on.
4501                     */
4502                    bcopy(mp->b_cont->b_rptr, dmp_cont->b_rptr, EUCSIZE);
4503                    dmp_cont->b_wptr += EUCSIZE;
4504                    dmp->b_cont = dmp_cont;
4505                    dmp->b_datap->db_type = M_CTL;
4506                    dmp_cont->b_datap->db_type = M_DATA;
```



```
4507                            riocp = (struct iocblk *)dmp->b_rptr;
4508                            riocp->ioc_count = EUCSIZE;
4509                            putnext(q, dmp);
4510
4511                            /*
4512                             * Now ACK the ioctl.
4513                             */
4514                            iocp->ioc_rval = 0;
4515                            miocack(q, mp, 0, 0);
4516                            return;
4517                    }
4518
4519            case EUC_WGET:
4520                    error = miocpullup(mp, sizeof (struct eucioc));
4521                    if (error != 0) {
4522                            miocnak(q, mp, 0, error);
4523                            return;
4524                    }
4525                    euciocp = (struct eucioc *)mp->b_cont->b_rptr;
4526                    cp_eucwioc(&tp->eucwioc, euciocp, EUCOUT);
4527                    iocp->ioc_rval = 0;
4528                    miocack(q, mp, EUCSIZE, 0);
4529                    return;
4530
4531            case CSDATA_SET:
4532                    error = miocpullup(mp, sizeof (ldterm_cs_data_user_t));
4533                    if (error != 0) {
4534                            miocnak(q, mp, 0, error);
4535                            return;
4536                    }
4537
4538                    csdp = (ldterm_cs_data_user_t *)mp->b_cont->b_rptr;
4539
4540                    /* Validate the codeset data provided. */
4541                    if (csdp->version > LDTERM_DATA_VERSION ||
4542                        csdp->codeset_type < LDTERM_CS_TYPE_MIN ||
4543                        csdp->codeset_type > LDTERM_CS_TYPE_MAX) {
4544                            miocnak(q, mp, 0, ERANGE);
4545                            return;
4546                    }
4547
4548                    if ((csdp->codeset_type == LDTERM_CS_TYPE_EUC &&
4549                        csdp->csinfo_num > LDTERM_CS_TYPE_EUC_MAX_SUBCS) ||
4550                        (csdp->codeset_type == LDTERM_CS_TYPE_PCCS &&
4551                        (csdp->csinfo_num < LDTERM_CS_TYPE_PCCS_MIN_SUBCS ||
4552                        csdp->csinfo_num > LDTERM_CS_TYPE_PCCS_MAX_SUBCS))) {
4553                            miocnak(q, mp, 0, ERANGE);
4554                            return;
4555                    }
4556
4557                    maxbytelen = maxscreenlen = 0;
4558                    if (csdp->codeset_type == LDTERM_CS_TYPE_EUC) {
4559                            for (i = 0; i < LDTERM_CS_TYPE_EUC_MAX_SUBCS; i++) {
4560                                    if (csdp->eucpc_data[i].byte_length >
4561                                        EUC_MAXW ||
4562                                        csdp->eucpc_data[i].screen_width >
4563                                        EUC_MAXW) {
4564                                            miocnak(q, mp, 0, ERANGE);
4565                                            return;
4566                                    }
4567
4568                                    if (csdp->eucpc_data[i].byte_length >
4569                                        maxbytelen)
4570                                            maxbytelen =
4571                                                csdp->eucpc_data[i].byte_length;
4572                                    if (csdp->eucpc_data[i].screen_width >
```

```
4573                                    maxscreenlen)
4574                                        maxscreenlen =
4575                                            csdp->eucpc_data[i].screen_width;
4576                            }
4577                            /* POSIX/C locale? */
4578                            if (maxbytelen == 0 && maxscreenlen == 0)
4579                                    maxbytelen = maxscreenlen = 1;
4580                    } else if (csdp->codeset_type == LDTERM_CS_TYPE_PCCS) {
4581                            for (i = 0; i < LDTERM_CS_MAX_CODESETS; i++) {
4582                                    if (csdp->eucpc_data[i].byte_length >
4583                                        LDTERM_CS_MAX_BYTE_LENGTH) {
4584                                        miocnak(q, mp, 0, ERANGE);
4585                                        return;
4586                                    }
4587                                    if (csdp->eucpc_data[i].byte_length >
4588                                        maxbytelen)
4589                                        maxbytelen =
4590                                            csdp->eucpc_data[i].byte_length;
4591                                    if (csdp->eucpc_data[i].screen_width >
4592                                        maxscreenlen)
4593                                        maxscreenlen =
4594                                            csdp->eucpc_data[i].screen_width;
4595                            }
4596                    } else if (csdp->codeset_type == LDTERM_CS_TYPE_UTF8) {
4597                            maxbytelen = 4;
4598                            maxscreenlen = 2;
4599                    }

4601                    locale_name_sz = 0;

4602                    if (csdp->locale_name) {
4603                    for (i = 0; i < MAXNAMELEN; i++)
4604                            if (csdp->locale_name[i] == '\0')
4605                                    break;
4606                    /*
4607                     * We cannot have any string that is not NULL byte
4608                     * terminated.
4609                     */
4610                    if (i >= MAXNAMELEN) {
4611                            miocnak(q, mp, 0, ERANGE);
4612                            return;
4613                    }

4615                    locale_name_sz = i + 1;
4616                    }

4617                    /*
4618                     * As the final check, if there was invalid codeset_type
4619                     * given, or invalid byte_length was specified, it's an error.
4620                     */
4621                    if (maxbytelen <= 0 || maxscreenlen <= 0) {
4622                            miocnak(q, mp, 0, ERANGE);
4623                            return;
4624                    }

4626                    /* Do the switching. */
4627                    tp->t_maxeuc = maxbytelen;
4628                    tp->t_state &= ~TS_MEUC;
4629                    if (maxbytelen > 1 || maxscreenlen > 1) {
4630                            if (!tp->t_eucp_mp) {
4631                                    if (!(tp->t_eucp_mp = allocb(_TTY_BUFSIZ,
4632                                        BPRI_HI))) {
4633                                        cmn_err(CE_WARN,
4634                                            "Can't allocate eucp_mp");
4635                                        miocnak(q, mp, 0, ENOSR);
4636                                        return;
```



```
4637                                    }
4638                                    /*
4639                                     * If there's junk in the canonical buffer,
4640                                     * then move the eucp pointer past it,
4641                                     * so we don't run off the beginning. This is
4642                                     * a total botch, but will hopefully keep
4643                                     * stuff from getting too messed up until
4644                                     * the user flushes this line!
4645                                     */
4646                                    if (tp->t_msglen) {
4647                                            tp->t_eucp = tp->t_eucp_mp->b_rptr;
4648                                            for (i = tp->t_msglen; i; i--)
4649                                                    *tp->t_eucp++ = 1;
4650                                    } else {
4651                                            tp->t_eucp = tp->t_eucp_mp->b_rptr;
4652                                    }
4653                            }

4655                            /*
4656                             * We only set TS_MEUC for a multibyte/multi-column
4657                             * codeset.
4658                             */
4659                            tp->t_state |= TS_MEUC;

4661                            tp->t_csdata.version = csdp->version;
4662                            tp->t_csdata.codeset_type = csdp->codeset_type;
4663                            tp->t_csdata.csinfo_num = csdp->csinfo_num;
4664                            bcopy(csdp->eucpc_data, tp->t_csdata.eucpc_data,
4665                                sizeof (ldterm_eucpc_data_t) *
4666                                LDTERM_CS_MAX_CODESETS);
4667                            tp->t_csmethods = cs_methods[csdp->codeset_type];

4669                            if (csdp->codeset_type == LDTERM_CS_TYPE_EUC) {
4670                                    tp->eucwioc.eucw[0] = 1;
4671                                    tp->eucwioc.scrw[0] = 1;

4673                                    tp->eucwioc.eucw[1] =
4674                                        csdp->eucpc_data[0].byte_length;
4675                                    tp->eucwioc.scrw[1] =
4676                                        csdp->eucpc_data[0].screen_width;

4678                                    tp->eucwioc.eucw[2] =
4679                                        csdp->eucpc_data[1].byte_length + 1;
4680                                    tp->eucwioc.scrw[2] =
4681                                        csdp->eucpc_data[1].screen_width;

4683                                    tp->eucwioc.eucw[3] =
4684                                        csdp->eucpc_data[2].byte_length + 1;
4685                                    tp->eucwioc.scrw[3] =
4686                                        csdp->eucpc_data[2].screen_width;
4687                            } else {
4688                                    /*
4689                                     * We are not going to use this data
4690                                     * structure. So, clear it. Also, stty(1) will
4691                                     * make use of the cleared tp->eucwioc when
4692                                     * it prints out codeset width setting.
4693                                     */
4694                                    bzero(&tp->eucwioc, EUCSIZE);
4695                            }
4696                    } else {
4697                            /*
4698                             * If this codeset is a single byte codeset that
4699                             * requires only single display column for all
4700                             * characters, we switch to default EUC codeset
4701                             * methods and data setting.
4702                             */
```

```
4704                        if (tp->t_eucp_mp) {
4705                                freemsg(tp->t_eucp_mp);
4706                                tp->t_eucp_mp = NULL;
4707                                tp->t_eucp = NULL;
4708                        }

4710                        bzero(&tp->eucwioc, EUCSIZE);
4711                        tp->eucwioc.eucw[0] = 1;
4712                        tp->eucwioc.scrw[0] = 1;
4713                        if (tp->t_csdata.locale_name != (char *)NULL) {
4714                                kmem_free(tp->t_csdata.locale_name,
4715                                    strlen(tp->t_csdata.locale_name) + 1);
4716                        }
4717                        tp->t_csdata = default_cs_data;
4718                        tp->t_csmethods = cs_methods[LDTERM_CS_TYPE_EUC];
4719                }

4721                /* Copy over locale_name. */
4722                if (tp->t_csdata.locale_name != (char *)NULL) {
4723                        kmem_free(tp->t_csdata.locale_name,
4724                            strlen(tp->t_csdata.locale_name) + 1);
4725                }
4726                if (locale_name_sz > 1) {
4727                        tp->t_csdata.locale_name = (char *)kmem_alloc(
4728                            locale_name_sz, KM_SLEEP);
4729                        (void) strcpy(tp->t_csdata.locale_name,
4730                            csdp->locale_name);
4731                } else {
4732                        tp->t_csdata.locale_name = (char *)NULL;
4733                }

4735                /*
4736                 * Now ACK the ioctl.
4737                 */
4738                iocp->ioc_rval = 0;
4739                miocack(q, mp, 0, 0);
4740                return;

4742        case CSDATA_GET:
4743                error = miocpullup(mp, sizeof (ldterm_cs_data_user_t));
4744                if (error != 0) {
4745                        miocnak(q, mp, 0, error);
4746                        return;
4747                }

4749                csdp = (ldterm_cs_data_user_t *)mp->b_cont->b_rptr;

4751                csdp->version = tp->t_csdata.version;
4752                csdp->codeset_type = tp->t_csdata.codeset_type;
4753                csdp->csinfo_num = tp->t_csdata.csinfo_num;
4754                csdp->pad = tp->t_csdata.pad;
4755                if (tp->t_csdata.locale_name) {
4756                        (void) strcpy(csdp->locale_name,
4757                            tp->t_csdata.locale_name);
4758                } else {
4759                        csdp->locale_name[0] = '\0';
4760                }
4761                bcopy(tp->t_csdata.eucpc_data, csdp->eucpc_data,
4762                    sizeof (ldterm_eucpc_data_t) * LDTERM_CS_MAX_CODESETS);
4763                /*
4764                 * If the codeset is an EUC codeset and if it has 2nd and/or
4765                 * 3rd supplementary codesets, we subtract one from each
4766                 * byte length of the supplementary codesets. This is
4767                 * because single shift characters, SS2 and SS3, are not
4768                 * included in the byte lengths in the user space.
```



```
4769                 */
4770                if (csdp->codeset_type == LDTERM_CS_TYPE_EUC) {
4771                        if (csdp->eucpc_data[1].byte_length)
4772                                csdp->eucpc_data[1].byte_length -= 1;
4773                        if (csdp->eucpc_data[2].byte_length)
4774                                csdp->eucpc_data[2].byte_length -= 1;
4775                }
4776                iocp->ioc_rval = 0;
4777                miocack(q, mp, sizeof (ldterm_cs_data_user_t), 0);
4778                return;

4780        case PTSSTTY:
4781                tp->t_state |= TS_ISPTSTTY;
4782                break;

4784        }

4786        putnext(q, mp);
4787 }
______unchanged_portion_omitted_
```