```
**********************************************************
   37046 Wed May  1 07:08:53 2019
new/usr/src/uts/common/io/wscons.c
10887 Missing void cast in wcuwsrv()
**********************************************************
/*
 * 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) 1987, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2019 Toomas Soome <tsoome@me.com>
 * Copyright 2019 Joyent, Inc.
 */

/*
 * "Workstation console" multiplexor driver for Sun.
 *
 * Sends output to the primary frame buffer using the PROM monitor;
 * gets input from a stream linked below us that is the "keyboard
 * driver", below which is linked the primary keyboard.
 */

/*
 * Locking Policy:
 * This module has a D_MTPERMOD inner perimeter which means STREAMS
 * only allows one thread to enter this module through STREAMS entry
 * points each time -- open() close() put() srv() qtimeout().
 * So for the most time we do not need locking in this module, but with
 * the following exceptions:
 *
 *   - wc shares three global variables (wc_dip, vc_active_console,
 *     vc_cons_user, vc_avl_root) with virtual console devname part
 *    (fs/dev/sdev_vtops.c) which get compiled into genunix.
 *
 *   - wc_modechg_cb() is a callback function which will triggered when
 *     framebuffer display mode is changed.
 *
 *   - vt_send_hotkeys() is triggered by timeout() which is not STREAMS MT
 *     safe.
 *
 * Based on the fact that virtual console devname part and wc_modechg_cb()
 * only do read access to the above mentioned shared four global variables,
 * It is safe to do locking this way:
 * 1) all read access to the four global variables in THIS WC MODULE do not
 *    need locking;
 * 2) all write access to the four global variables in THIS WC MODULE must
 *    hold vc_lock;
 * 3) any access to the four global variables in either DEVNAME PART or the
```

```
 *    CALLBACK must hold vc_lock;
 * 4) other global variables which are only shared in this wc module and only
 *    accessible through STREAMS entry points such as "vc_last_console",
 *    "vc_inuse_max_minor", "vc_target_console" and "vc_waitactive_list"
 *    do not need explict locking.
 *
 * wc_modechg_cb() does read access to vc_state_t::vc_flags,
 * vc_state_t::vc_state_lock is used to protect concurrently accesses to
 * vc_state_t::vc_flags which may happen from both through STREAMS entry
 * points and wc_modechg_cb().
 * Since wc_modechg_cb() only does read access to vc_state_t::vc_flags,
 * The other parts of wc module (except wc_modechg_cb()) only has to hold
 * vc_state_t::vc_flags when writing to vc_state_t::vc_flags.
 *
 * vt_send_hotkeys() could access vt_pending_vtno at the same time with
 * the rest of wc module, vt_pending_vtno_lock is used to protect
 * vt_pending_vtno.
 *
 * Lock order: vc_lock -> vc_state_t::vc_state_lock.
 * No overlap between vc_lock and vt_pending_vtno_lock.
 */

#include <sys/types.h>
#include <sys/param.h>
#include <sys/signal.h>
#include <sys/cred.h>
#include <sys/vnode.h>
#include <sys/termios.h>
#include <sys/termio.h>
#include <sys/ttold.h>
#include <sys/stropts.h>
#include <sys/stream.h>
#include <sys/strsun.h>
#include <sys/tty.h>
#include <sys/buf.h>
#include <sys/uio.h>
#include <sys/stat.h>
#include <sys/sysmacros.h>
#include <sys/errno.h>
#include <sys/proc.h>
#include <sys/procset.h>
#include <sys/fault.h>
#include <sys/siginfo.h>
#include <sys/debug.h>
#include <sys/session.h>
#include <sys/kmem.h>
#include <sys/cpuvar.h>
#include <sys/kbio.h>
#include <sys/strredir.h>
#include <sys/fs/snode.h>
#include <sys/consdev.h>
#include <sys/conf.h>
#include <sys/cmn_err.h>
#include <sys/console.h>
#include <sys/promif.h>
#include <sys/note.h>
#include <sys/polled_io.h>
#include <sys/systm.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/sunndi.h>
#include <sys/esunddi.h>
#include <sys/sunldi.h>
#include <sys/debug.h>
#include <sys/console.h>
#include <sys/ddi_impldefs.h>
```

```
#include <sys/policy.h>
#include <sys/modctl.h>
#include <sys/tem.h>
#include <sys/wscons.h>
#include <sys/vt_impl.h>

/* streams stuff */
_NOTE(SCHEME_PROTECTS_DATA("Unshared data", copyreq))
_NOTE(SCHEME_PROTECTS_DATA("Unshared data", copyresp))
_NOTE(SCHEME_PROTECTS_DATA("Unshared data", datab))
_NOTE(SCHEME_PROTECTS_DATA("Unshared data", iocblk))
_NOTE(SCHEME_PROTECTS_DATA("Unshared data", msgb))
_NOTE(SCHEME_PROTECTS_DATA("Unshared data", queue))

#define MINLINES        10
#define MAXLINES        48
#define LOSCREENLINES   34
#define HISCREENLINES   48

#define MINCOLS         10
#define MAXCOLS         120
#define LOSCREENCOLS    80
#define HISCREENCOLS    120

struct wscons_state {
        dev_t   wc_dev;                 /* major/minor for this device */
#ifdef _HAVE_TEM_FIRMWARE
        int     wc_defer_output;        /* set if output device is "slow" */
#endif /* _HAVE_TEM_FIRMWARE */
        queue_t *wc_kbdqueue;           /* "console keyboard" device queue */
                                        /* below us */
        cons_polledio_t         wc_polledio; /* polled I/O function pointers */
        cons_polledio_t         *wc_kb_polledio; /* keyboard's polledio */
        unsigned int    wc_kb_getpolledio_id; /* id for kb CONSOPENPOLLEDIO */
        queue_t *wc_pending_wq;
        mblk_t  *wc_pending_link;       /* I_PLINK pending for kb polledio */
} wscons;
```
______unchanged_portion_omitted_

```
/*
 * Service procedure for upper write queue.
 * We need to have service procedure to make sure the keyboard events
 * are queued up for screen output and are not dependant on the screen
 * updates.
 */
static int
wcuwsrv(queue_t *q)
{
        vc_state_t *pvc = (vc_state_t *)q->q_ptr;
        tem_vt_state_t ptem = NULL;
        mblk_t *mp;
        ssize_t cc;

        while ((mp = getq(q)) != NULL) {
                /*
                 * If we're waiting for something to happen (delay timeout to
                 * expire, current transmission to finish, output to be
                 * restarted, output to finish draining), don't grab anything
                 * new.
                 */
                if (pvc->vc_flags & (WCS_DELAY|WCS_BUSY|WCS_STOPPED)) {
                        (void) putbq(q, mp);
                        putbq(q, mp);
                        return (0);
                }
```

```
                switch (mp->b_datap->db_type) {
                default:        /* drop unknown type */
                        freemsg(mp);
                        continue;

                case M_IOCTL:
                        wcioctl(q, mp);
                        continue;

                case M_DELAY:
                        /*
                         * Arrange for "wcrstrt" to be called when the
                         * delay expires; it will turn WCS_DELAY off.
                         */
                        if (pvc->vc_timeoutid != 0)
                                (void) quntimeout(q, pvc->vc_timeoutid);
                        pvc->vc_timeoutid = qtimeout(q, wcrstrt, pvc,
                            (clock_t)(*(unsigned char *)mp->b_rptr + 6));

                        mutex_enter(&pvc->vc_state_lock);
                        pvc->vc_flags |= WCS_DELAY;
                        mutex_exit(&pvc->vc_state_lock);

                        freemsg(mp);
                        continue;

                case M_DATA:
                        break;
                }

                if ((cc = mp->b_wptr - mp->b_rptr) == 0) {
                        freemsg(mp);
                        continue;
                }

#ifdef _HAVE_TEM_FIRMWARE
                if (consmode == CONS_KFB) {
#endif /* _HAVE_TEM_FIRMWARE */
                        ptem = wc_get_screen_tem(pvc);

                        if (ptem == NULL) {
                                freemsg(mp);
                                continue;
                        }

                        for (mblk_t *nbp = mp; nbp != NULL; nbp = nbp->b_cont) {
                                cc = nbp->b_wptr - nbp->b_rptr;

                                if (cc <= 0)
                                        continue;

                                tem_write(ptem, nbp->b_rptr, cc, kcred);
                        }
                        freemsg(mp);
#ifdef _HAVE_TEM_FIRMWARE
                        continue;
                }

                /* consmode = CONS_FW */
                if (pvc->vc_minor != 0) {
                        freemsg(mp);
                        continue;
                }

                /*
                 * Direct output to the frame buffer if this device
                 * is not the "hardware" console.
                 */
                if (wscons.wc_defer_output) {
                        mutex_enter(&pvc->vc_state_lock);
                        pvc->vc_flags |= WCS_BUSY;
                        mutex_exit(&pvc->vc_state_lock);

                        pvc->vc_pendc = -1;

                        for (mblk_t *nbp = mp; nbp != NULL; nbp = nbp->b_cont) {
                                cc = nbp->b_wptr - nbp->b_rptr;

                                if (cc <= 0)
                                        continue;

                                console_puts((const char *)nbp->b_rptr, cc);
                        }
                        freemsg(mp);
                        mutex_enter(&pvc->vc_state_lock);
                        pvc->vc_flags &= ~WCS_BUSY;
                        mutex_exit(&pvc->vc_state_lock);
                        continue;
                }
                for (boolean_t done = B_FALSE; done != B_TRUE; ) {
                        int c;

                        c = *mp->b_rptr++;
                        cc--;
                        if (prom_mayput((char)c) != 0) {

                                mutex_enter(&pvc->vc_state_lock);
                                pvc->vc_flags |= WCS_BUSY;
                                mutex_exit(&pvc->vc_state_lock);

                                pvc->vc_pendc = c;
                                if (pvc->vc_timeoutid != 0)
                                        (void) quntimeout(q,
                                            pvc->vc_timeoutid);
                                pvc->vc_timeoutid = qtimeout(q, wcopoll,
                                    pvc, 1);
                                if (mp != NULL) {
                                        /* not done with this message yet */
                                        (void) putbq(q, mp);
                                        return (0);
                                }
                                break;
                        }
                        while (cc <= 0) {
                                mblk_t *nbp = mp;
                                mp = mp->b_cont;
                                freeb(nbp);
                                if (mp == NULL) {
                                        done = B_TRUE;
                                        break;
                                }
                                /* LINTED E_PTRDIFF_OVERFLOW */
                                cc = mp->b_wptr - mp->b_rptr;
                        }
                }
#endif /* _HAVE_TEM_FIRMWARE */
        }
        return (0);
}
```
______*__unchanged_portion_omitted_*__