****
*   51582 Thu Dec 17 22:54:47 2015*
new/usr/src/uts/common/fs/fifofs/fifovnops.c
6474 event ports are broken with FIFOs
****
______*unchanged_portion_omitted_*

```
static inline int
fifo_ioctl_getpeercred(fifonode_t *fnp, intptr_t arg, int mode)
{
        k_peercred_t *kp = (k_peercred_t *)arg;

        if (mode == FKIOCTL && fnp->fn_pcredp != NULL) {
                crhold(fnp->fn_pcredp);
                kp->pc_cr = fnp->fn_pcredp;
                kp->pc_cpid = fnp->fn_cpid;
                return (0);
        } else {
                return (ENOTSUP);
        }
}

#endif /* ! codereview */
static int
fifo_fastioctl(vnode_t *vp, int cmd, intptr_t arg, int mode,
        cred_t *cr, int *rvalp)
{
        fifonode_t      *fnp            = VTOF(vp);
        fifonode_t      *fn_dest;
        int             error           = 0;
        fifolock_t      *fn_lock        = fnp->fn_lock;
        int             cnt;

        /*
         * tty operations not allowed
         */
        if (((cmd & IOCTYPE) == LDIOC) ||
            ((cmd & IOCTYPE) == tIOC) ||
            ((cmd & IOCTYPE) == TIOC)) {
                return (EINVAL);
        }

        mutex_enter(&fn_lock->flk_lock);

        if (!(fnp->fn_flag & FIFOFAST)) {
                goto stream_mode;
        }

        switch (cmd) {

        /*
         * Things we can't handle
         * These will switch us to streams mode.
         */
        default:
        case I_STR:
        case I_SRDOPT:
        case I_PUSH:
        case I_FDINSERT:
        case I_SENDFD:
        case I_RECVFD:
        case I_E_RECVFD:
        case I_ATMARK:
        case I_CKBAND:
        case I_GETBAND:
        case I_SWROPT:
```

```
                goto turn_fastoff;

        /*
         * Things that don't do damage
         * These things don't adjust the state of the
         * stream head (i_setcltime does, but we don't care)
         */
        case I_FIND:
        case I_GETSIG:
        case FIONBIO:
        case FIOASYNC:
        case I_GRDOPT:  /* probably should not get this, but no harm */
        case I_GWROPT:
        case I_LIST:
        case I_SETCLTIME:
        case I_GETCLTIME:
                mutex_exit(&fn_lock->flk_lock);
                return (strioctl(vp, cmd, arg, mode, U_TO_K, cr, rvalp));

        case I_CANPUT:
                /*
                 * We can only handle normal band canputs.
                 * XXX : We could just always go to stream mode; after all
                 * canput is a streams semantics type thing
                 */
                if (arg != 0) {
                        goto turn_fastoff;
                }
                *rvalp = (fnp->fn_dest->fn_count < Fifohiwat) ? 1 : 0;
                mutex_exit(&fn_lock->flk_lock);
                return (0);

        case I_NREAD:
                /*
                 * This may seem a bit silly for non-streams semantics,
                 * (After all, if they really want a message, they'll
                 * probably use getmsg() anyway). but it doesn't hurt
                 */
                error = copyout((caddr_t)&fnp->fn_count, (caddr_t)arg,
                    sizeof (cnt));
                if (error == 0) {
                        *rvalp = (fnp->fn_count == 0) ? 0 : 1;
                }
                break;

        case FIORDCHK:
                *rvalp = fnp->fn_count;
                break;

        case I_PEEK:
        {
                STRUCT_DECL(strpeek, strpeek);
                struct uio      uio;
                struct iovec    iov;
                int             count;
                mblk_t          *bp;
                int             len;

                STRUCT_INIT(strpeek, mode);

                if (fnp->fn_count == 0) {
                        *rvalp = 0;
                        break;
                }

                error = copyin((caddr_t)arg, STRUCT_BUF(strpeek),
		    STRUCT_SIZE(strpeek));
		if (error)
			break;

		/*
		 * can't have any high priority message when in fast mode
		 */
		if (STRUCT_FGET(strpeek, flags) & RS_HIPRI) {
			*rvalp = 0;
			break;
		}

		len = STRUCT_FGET(strpeek, databuf.maxlen);
		if (len <= 0) {
			STRUCT_FSET(strpeek, databuf.len, len);
		} else {
			iov.iov_base = STRUCT_FGETP(strpeek, databuf.buf);
			iov.iov_len = len;
			uio.uio_iov = &iov;
			uio.uio_iovcnt = 1;
			uio.uio_loffset = 0;
			uio.uio_segflg = UIO_USERSPACE;
			uio.uio_fmode = 0;
			/* For pipes copy should not bypass cache */
			uio.uio_extflg = UIO_COPY_CACHED;
			uio.uio_resid = iov.iov_len;
			count = fnp->fn_count;
			bp = fnp->fn_mp;
			while (count > 0 && uio.uio_resid) {
				cnt = MIN(uio.uio_resid, MBLKL(bp));
				if ((error = uiomove((char *)bp->b_rptr, cnt,
				    UIO_READ, &uio)) != 0) {
					break;
				}
				count -= cnt;
				bp = bp->b_cont;
			}
			STRUCT_FSET(strpeek, databuf.len, len - uio.uio_resid);
		}
		STRUCT_FSET(strpeek, flags, 0);
		STRUCT_FSET(strpeek, ctlbuf.len, -1);

		error = copyout(STRUCT_BUF(strpeek), (caddr_t)arg,
		    STRUCT_SIZE(strpeek));
		if (error == 0 && len >= 0)
			*rvalp = 1;
		break;
	}

	case FIONREAD:
		/*
		 * let user know total number of bytes in message queue
		 */
		error = copyout((caddr_t)&fnp->fn_count, (caddr_t)arg,
		    sizeof (fnp->fn_count));
		if (error == 0)
			*rvalp = 0;
		break;

	case I_SETSIG:
		/*
		 * let streams set up the signal masking for us
		 * we just check to see if it's set
		 * XXX : this interface should not be visible
		 *  i.e. STREAM's framework is exposed.
		 */
```

```
		error = strioctl(vp, cmd, arg, mode, U_TO_K, cr, rvalp);
		if (vp->v_stream->sd_sigflags & (S_INPUT|S_RDNORM|S_WRNORM))
			fnp->fn_flag |= FIFOSETSIG;
		else
			fnp->fn_flag &= ~FIFOSETSIG;
		break;

	case I_FLUSH:
		/*
		 * flush them message queues
		 */
		if (arg & ~FLUSHRW) {
			error = EINVAL;
			break;
		}
		if (arg & FLUSHR) {
			fifo_fastflush(fnp);
		}
		fn_dest = fnp->fn_dest;
		if ((arg & FLUSHW)) {
			fifo_fastflush(fn_dest);
		}
		/*
		 * wake up any sleeping readers or writers
		 * (waking readers probably doesn't make sense, but it
		 *  doesn't hurt; i.e. we just got rid of all the data
		 *  what's to read ?)
		 */
		if (fn_dest->fn_flag & (FIFOWANTW | FIFOWANTR)) {
			fn_dest->fn_flag &= ~(FIFOWANTW | FIFOWANTR);
			cv_broadcast(&fn_dest->fn_wait_cv);
		}
		*rvalp = 0;
		break;

	/*
	 * Since no band data can ever get on a fifo in fast mode
	 * just return 0.
	 */
	case I_FLUSHBAND:
		error = 0;
		*rvalp = 0;
		break;

	case _I_GETPEERCRED:
		error = fifo_ioctl_getpeercred(fnp, arg, mode);
		break;

#endif /* ! codereview */
	/*
	 * invalid calls for stream head or fifos
	 */

	case I_POP:		/* shouldn't happen */
	case I_LOOK:
	case I_LINK:
	case I_PLINK:
	case I_UNLINK:
	case I_PUNLINK:

	/*
	 * more invalid tty type of ioctls
	 */

	case SRIOCSREDIR:
	case SRIOCISREDIR:
```

```
                error = EINVAL;
                break;

        }
        mutex_exit(&fn_lock->flk_lock);
        return (error);

turn_fastoff:
        fifo_fastoff(fnp);

stream_mode:
        /*
         * streams mode
         */
        mutex_exit(&fn_lock->flk_lock);
        return (fifo_strioctl(vp, cmd, arg, mode, cr, rvalp));

}

/*
 * FIFO is in STREAMS mode; STREAMS framework does most of the work.
 */
static int
fifo_strioctl(vnode_t *vp, int cmd, intptr_t arg, int mode,
        cred_t *cr, int *rvalp)
{
        fifonode_t      *fnp = VTOF(vp);
        int             error;
        fifolock_t      *fn_lock;

        if (cmd == _I_GETPEERCRED)
                return (fifo_ioctl_getpeercred(fnp, arg, mode));
        if (cmd == _I_GETPEERCRED) {
                if (mode == FKIOCTL && fnp->fn_pcredp != NULL) {
                        k_peercred_t *kp = (k_peercred_t *)arg;
                        crhold(fnp->fn_pcredp);
                        kp->pc_cr = fnp->fn_pcredp;
                        kp->pc_cpid = fnp->fn_cpid;
                        return (0);
                } else {
                        return (ENOTSUP);
                }
        }

        error = strioctl(vp, cmd, arg, mode, U_TO_K, cr, rvalp);

        switch (cmd) {
        /*
         * The FIFOSEND flag is set to inform other processes that a file
         * descriptor is pending at the stream head of this pipe.
         * The flag is cleared and the sending process is awoken when
         * this process has completed receiving the file descriptor.
         * XXX This could become out of sync if the process does I_SENDFDs
         * and opens on connld attached to the same pipe.
         */
        case I_RECVFD:
        case I_E_RECVFD:
                if (error == 0) {
                        fn_lock = fnp->fn_lock;
                        mutex_enter(&fn_lock->flk_lock);
                        if (fnp->fn_flag & FIFOSEND) {
                                fnp->fn_flag &= ~FIFOSEND;
                                cv_broadcast(&fnp->fn_dest->fn_wait_cv);
                        }
                        mutex_exit(&fn_lock->flk_lock);
                }
```

```
                break;
        default:
                break;
        }

        return (error);
}
```

______*unchanged_portion_omitted_*