```
**********************************************************
   51675 Mon Jun 17 07:28:02 2019
new/usr/src/uts/common/fs/fifofs/fifovnops.c
6474 getupeercred causes spurious event port wakeups on FIFOs
**********************************************************
______unchanged_portion_omitted_

1130 static inline int
1131 fifo_ioctl_getpeercred(fifonode_t *fnp, intptr_t arg, int mode)
1132 {
1133         k_peercred_t *kp = (k_peercred_t *)arg;

1135         if (mode == FKIOCTL && fnp->fn_pcredp != NULL) {
1136                 crhold(fnp->fn_pcredp);
1137                 kp->pc_cr = fnp->fn_pcredp;
1138                 kp->pc_cpid = fnp->fn_cpid;
1139                 return (0);
1140         } else {
1141                 return (ENOTSUP);
1142         }
1143 }

1145 static int
1146 fifo_fastioctl(vnode_t *vp, int cmd, intptr_t arg, int mode, cred_t *cr,
1147     int *rvalp)
1148 {
1149         fifonode_t      *fnp            = VTOF(vp);
1150         fifonode_t      *fn_dest;
1151         int             error           = 0;
1152         fifolock_t      *fn_lock        = fnp->fn_lock;
1153         int             cnt;

1155         /*
1156          * tty operations not allowed
1157          */
1158         if (((cmd & IOCTYPE) == LDIOC) ||
1159             ((cmd & IOCTYPE) == tIOC) ||
1160             ((cmd & IOCTYPE) == TIOC)) {
1161                 return (EINVAL);
1162         }

1164         mutex_enter(&fn_lock->flk_lock);

1166         if (!(fnp->fn_flag & FIFOFAST)) {
1167                 goto stream_mode;
1168         }

1170         switch (cmd) {

1172         /*
1173          * Things we can't handle
1174          * These will switch us to streams mode.
1175          */
1176         default:
1177         case I_STR:
1178         case I_SRDOPT:
1179         case I_PUSH:
1180         case I_FDINSERT:
1181         case I_SENDFD:
1182         case I_RECVFD:
1183         case I_E_RECVFD:
1184         case I_ATMARK:
1185         case I_CKBAND:
1186         case I_GETBAND:
1187         case I_SWROPT:
1188                 goto turn_fastoff;
```



```
1190         /*
1191          * Things that don't do damage
1192          * These things don't adjust the state of the
1193          * stream head (i_setcltime does, but we don't care)
1194          */
1195         case I_FIND:
1196         case I_GETSIG:
1197         case FIONBIO:
1198         case FIOASYNC:
1199         case I_GRDOPT:  /* probably should not get this, but no harm */
1200         case I_GWROPT:
1201         case I_LIST:
1202         case I_SETCLTIME:
1203         case I_GETCLTIME:
1204                 mutex_exit(&fn_lock->flk_lock);
1205                 return (strioctl(vp, cmd, arg, mode, U_TO_K, cr, rvalp));

1207         case I_CANPUT:
1208                 /*
1209                  * We can only handle normal band canputs.
1210                  * XXX : We could just always go to stream mode; after all
1211                  * canput is a streams semantics type thing
1212                  */
1213                 if (arg != 0) {
1214                         goto turn_fastoff;
1215                 }
1216                 *rvalp = (fnp->fn_dest->fn_count < Fifohiwat) ? 1 : 0;
1217                 mutex_exit(&fn_lock->flk_lock);
1218                 return (0);

1220         case I_NREAD:
1221                 /*
1222                  * This may seem a bit silly for non-streams semantics,
1223                  * (After all, if they really want a message, they'll
1224                  * probably use getmsg() anyway). but it doesn't hurt
1225                  */
1226                 error = copyout((caddr_t)&fnp->fn_count, (caddr_t)arg,
1227                     sizeof (cnt));
1228                 if (error == 0) {
1229                         *rvalp = (fnp->fn_count == 0) ? 0 : 1;
1230                 }
1231                 break;

1233         case FIORDCHK:
1234                 *rvalp = fnp->fn_count;
1235                 break;

1237         case I_PEEK:
1238         {
1239                 STRUCT_DECL(strpeek, strpeek);
1240                 struct uio      uio;
1241                 struct iovec    iov;
1242                 int             count;
1243                 mblk_t          *bp;
1244                 int             len;

1246                 STRUCT_INIT(strpeek, mode);

1248                 if (fnp->fn_count == 0) {
1249                         *rvalp = 0;
1250                         break;
1251                 }

1253                 error = copyin((caddr_t)arg, STRUCT_BUF(strpeek),
1254                     STRUCT_SIZE(strpeek));
```

```
1255                if (error)
1256                        break;

1258                /*
1259                 * can't have any high priority message when in fast mode
1260                 */
1261                if (STRUCT_FGET(strpeek, flags) & RS_HIPRI) {
1262                        *rvalp = 0;
1263                        break;
1264                }

1266                len = STRUCT_FGET(strpeek, databuf.maxlen);
1267                if (len <= 0) {
1268                        STRUCT_FSET(strpeek, databuf.len, len);
1269                } else {
1270                        iov.iov_base = STRUCT_FGETP(strpeek, databuf.buf);
1271                        iov.iov_len = len;
1272                        uio.uio_iov = &iov;
1273                        uio.uio_iovcnt = 1;
1274                        uio.uio_loffset = 0;
1275                        uio.uio_segflg = UIO_USERSPACE;
1276                        uio.uio_fmode = 0;
1277                        /* For pipes copy should not bypass cache */
1278                        uio.uio_extflg = UIO_COPY_CACHED;
1279                        uio.uio_resid = iov.iov_len;
1280                        count = fnp->fn_count;
1281                        bp = fnp->fn_mp;
1282                        while (count > 0 && uio.uio_resid) {
1283                                cnt = MIN(uio.uio_resid, MBLKL(bp));
1284                                if ((error = uiomove((char *)bp->b_rptr, cnt,
1285                                    UIO_READ, &uio)) != 0) {
1286                                        break;
1287                                }
1288                                count -= cnt;
1289                                bp = bp->b_cont;
1290                        }
1291                        STRUCT_FSET(strpeek, databuf.len, len - uio.uio_resid);
1292                }
1293                STRUCT_FSET(strpeek, flags, 0);
1294                STRUCT_FSET(strpeek, ctlbuf.len, -1);

1296                error = copyout(STRUCT_BUF(strpeek), (caddr_t)arg,
1297                    STRUCT_SIZE(strpeek));
1298                if (error == 0 && len >= 0)
1299                        *rvalp = 1;
1300                break;
1301        }

1303        case FIONREAD:
1304                /*
1305                 * let user know total number of bytes in message queue
1306                 */
1307                error = copyout((caddr_t)&fnp->fn_count, (caddr_t)arg,
1308                    sizeof (fnp->fn_count));
1309                if (error == 0)
1310                        *rvalp = 0;
1311                break;

1313        case I_SETSIG:
1314                /*
1315                 * let streams set up the signal masking for us
1316                 * we just check to see if it's set
1317                 * XXX : this interface should not be visible
1318                 *  i.e. STREAM's framework is exposed.
1319                 */
1320                error = strioctl(vp, cmd, arg, mode, U_TO_K, cr, rvalp);
```



```
1321                if (vp->v_stream->sd_sigflags & (S_INPUT|S_RDNORM|S_WRNORM))
1322                        fnp->fn_flag |= FIFOSETSIG;
1323                else
1324                        fnp->fn_flag &= ~FIFOSETSIG;
1325                break;

1327        case I_FLUSH:
1328                /*
1329                 * flush them message queues
1330                 */
1331                if (arg & ~FLUSHRW) {
1332                        error = EINVAL;
1333                        break;
1334                }
1335                if (arg & FLUSHR) {
1336                        fifo_fastflush(fnp);
1337                }
1338                fn_dest = fnp->fn_dest;
1339                if ((arg & FLUSHW)) {
1340                        fifo_fastflush(fn_dest);
1341                }
1342                /*
1343                 * wake up any sleeping readers or writers
1344                 * (waking readers probably doesn't make sense, but it
1345                 *  doesn't hurt; i.e. we just got rid of all the data
1346                 *  what's to read ?)
1347                 */
1348                if (fn_dest->fn_flag & (FIFOWANTW | FIFOWANTR)) {
1349                        fn_dest->fn_flag &= ~(FIFOWANTW | FIFOWANTR);
1350                        cv_broadcast(&fn_dest->fn_wait_cv);
1351                }
1352                *rvalp = 0;
1353                break;

1355        /*
1356         * Since no band data can ever get on a fifo in fast mode
1357         * just return 0.
1358         */
1359        case I_FLUSHBAND:
1360                error = 0;
1361                *rvalp = 0;
1362                break;

1364        case _I_GETPEERCRED:
1365                error = fifo_ioctl_getpeercred(fnp, arg, mode);
1366                break;

1368        /*
1369         * invalid calls for stream head or fifos
1370         */

1372        case I_POP:             /* shouldn't happen */
1373        case I_LOOK:
1374        case I_LINK:
1375        case I_PLINK:
1376        case I_UNLINK:
1377        case I_PUNLINK:

1379        /*
1380         * more invalid tty type of ioctls
1381         */

1383        case SRIOCSREDIR:
1384        case SRIOCISREDIR:
1385                error = EINVAL;
1386                break;
```

```
1388         }
1389         mutex_exit(&fn_lock->flk_lock);
1390         return (error);

1392 turn_fastoff:
1393         fifo_fastoff(fnp);

1395 stream_mode:
1396         /*
1397          * streams mode
1398          */
1399         mutex_exit(&fn_lock->flk_lock);
1400         return (fifo_strioctl(vp, cmd, arg, mode, cr, rvalp));

1402 }

1404 /*
1405  * FIFO is in STREAMS mode; STREAMS framework does most of the work.
1406  */
1407 static int
1408 fifo_strioctl(vnode_t *vp, int cmd, intptr_t arg, int mode, cred_t *cr,
1409     int *rvalp)
1410 {
1411         fifonode_t      *fnp = VTOF(vp);
1412         int             error;
1413         fifolock_t      *fn_lock;

1415         if (cmd == _I_GETPEERCRED)
1416                 return (fifo_ioctl_getpeercred(fnp, arg, mode));
1396         if (cmd == _I_GETPEERCRED) {
1397                 if (mode == FKIOCTL && fnp->fn_pcredp != NULL) {
1398                         k_peercred_t *kp = (k_peercred_t *)arg;
1399                         crhold(fnp->fn_pcredp);
1400                         kp->pc_cr = fnp->fn_pcredp;
1401                         kp->pc_cpid = fnp->fn_cpid;
1402                         return (0);
1403                 } else {
1404                         return (ENOTSUP);
1405                 }
1406         }

1418         error = strioctl(vp, cmd, arg, mode, U_TO_K, cr, rvalp);

1420         switch (cmd) {
1421         /*
1422          * The FIFOSEND flag is set to inform other processes that a file
1423          * descriptor is pending at the stream head of this pipe.
1424          * The flag is cleared and the sending process is awoken when
1425          * this process has completed receiving the file descriptor.
1426          * XXX This could become out of sync if the process does I_SENDFDs
1427          * and opens on connld attached to the same pipe.
1428          */
1429         case I_RECVFD:
1430         case I_E_RECVFD:
1431                 if (error == 0) {
1432                         fn_lock = fnp->fn_lock;
1433                         mutex_enter(&fn_lock->flk_lock);
1434                         if (fnp->fn_flag & FIFOSEND) {
1435                                 fnp->fn_flag &= ~FIFOSEND;
1436                                 cv_broadcast(&fnp->fn_dest->fn_wait_cv);
1437                         }
1438                         mutex_exit(&fn_lock->flk_lock);
1439                 }
1440                 break;
1441         default:
```



```
1442                 break;
1443         }

1445         return (error);
1446 }
```

_____***unchanged_portion_omitted_***