```
**********************************************************
   61202 Wed Jul 23 20:11:14 2014
new/usr/src/uts/common/fs/sockfs/sockcommon_subr.c
5026 intra-node/inter-zone networking doesn't always deliver SIGPOLL
**********************************************************
```

```
/*
 * 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) 2008, 2010, Oracle and/or its affiliates. All rights reserved.
 */
/*
 * Copyright 2014, OmniTI Computer Consulting, Inc. All rights reserved.
 */

#include <sys/types.h>
#include <sys/param.h>
#include <sys/signal.h>
#include <sys/cmn_err.h>

#include <sys/stropts.h>
#include <sys/socket.h>
#include <sys/socketvar.h>
#include <sys/sockio.h>
#include <sys/strsubr.h>
#include <sys/strsun.h>
#include <sys/atomic.h>
#include <sys/tihdr.h>

#include <fs/sockfs/sockcommon.h>
#include <fs/sockfs/sockfilter_impl.h>
#include <fs/sockfs/socktpi.h>
#include <fs/sockfs/sodirect.h>
#include <sys/ddi.h>
#include <inet/ip.h>
#include <sys/time.h>
#include <sys/cmn_err.h>

#ifdef SOCK_TEST
extern int do_useracc;
extern clock_t sock_test_timelimit;
#endif /* SOCK_TEST */

#define MBLK_PULL_LEN 64
uint32_t so_mblk_pull_len = MBLK_PULL_LEN;

#ifdef DEBUG
boolean_t so_debug_length = B_FALSE;
```



```
static boolean_t so_check_length(sonode_t *so);
#endif

static int
so_acceptq_dequeue_locked(struct sonode *so, boolean_t dontblock,
    struct sonode **nsop)
{
        struct sonode *nso = NULL;

        *nsop = NULL;
        ASSERT(MUTEX_HELD(&so->so_acceptq_lock));
        while ((nso = list_remove_head(&so->so_acceptq_list)) == NULL) {
                /*
                 * No need to check so_error here, because it is not
                 * possible for a listening socket to be reset or otherwise
                 * disconnected.
                 *
                 * So now we just need check if it's ok to wait.
                 */
                if (dontblock)
                        return (EWOULDBLOCK);
                if (so->so_state & (SS_CLOSING | SS_FALLBACK_PENDING))
                        return (EINTR);

                if (cv_wait_sig_swap(&so->so_acceptq_cv,
                    &so->so_acceptq_lock) == 0)
                        return (EINTR);
        }

        ASSERT(nso != NULL);
        ASSERT(so->so_acceptq_len > 0);
        so->so_acceptq_len--;
        nso->so_listener = NULL;

        *nsop = nso;

        return (0);
}
```

*unchanged_portion_omitted_*

```
void
socket_sendsig(struct sonode *so, int event)
{
        proc_t *proc;

        ASSERT(MUTEX_HELD(&so->so_lock));

        if (so->so_pgrp == 0 || (!(so->so_state & SS_ASYNC) &&
            event != SOCKETSIG_URG)) {
                return;
        }

        dprint(3, ("sending sig %d to %d\n", event, so->so_pgrp));

        if (so->so_pgrp > 0) {
                /*
                 * XXX This unfortunately still generates
                 * a signal when a fd is closed but
                 * the proc is active.
                 */
                mutex_enter(&pidlock);
                /*
                 * Even if the thread started in another zone, we're receiving
                 * on behalf of this socket's zone, so find the proc using the
                 * socket's zone ID.
                 */
```

```
407                 proc = prfind_zone(so->so_pgrp, so->so_zoneid);
399                 proc = prfind(so->so_pgrp);
408                 if (proc == NULL) {
409                         mutex_exit(&pidlock);
410                         return;
411                 }
412                 mutex_enter(&proc->p_lock);
413                 mutex_exit(&pidlock);
414                 socket_sigproc(proc, event);
415                 mutex_exit(&proc->p_lock);
416         } else {
417                 /*
418                  * Send to process group. Hold pidlock across
419                  * calls to socket_sigproc().
420                  */
421                 pid_t pgrp = -so->so_pgrp;

423                 mutex_enter(&pidlock);
424                 /*
425                  * Even if the thread started in another zone, we're receiving
426                  * on behalf of this socket's zone, so find the pgrp using the
427                  * socket's zone ID.
428                  */
429                 proc = pgfind_zone(pgrp, so->so_zoneid);
416                 proc = pgfind(pgrp);
430                 while (proc != NULL) {
431                         mutex_enter(&proc->p_lock);
432                         socket_sigproc(proc, event);
433                         mutex_exit(&proc->p_lock);
434                         proc = proc->p_pglink;
435                 }
436                 mutex_exit(&pidlock);
437         }
438 }
```

*unchanged_portion_omitted_*