```
**********************************************************
   23565 Sun Nov  2 01:27:47 2014
new/usr/src/lib/libc/port/threads/sigaction.c
5281 incorrect realtime signal delivery
**********************************************************
```

```
/*
 * 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 2014 Ryan Zezeski.  All rights reserved.
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "lint.h"
#include <sys/feature_tests.h>
/*
 * setcontext() really can return, if UC_CPU is not specified.
 * Make the compiler shut up about it.
 */
#if defined(__NORETURN)
#undef  __NORETURN
#endif
#define __NORETURN
#include "thr_uberdata.h"
#include "asyncio.h"
#include <signal.h>
#include <siginfo.h>
#include <sys/systm.h>

/* maskable signals */
const sigset_t maskset = {MASKSET0, MASKSET1, MASKSET2, MASKSET3};

/*
 * Return true if the valid signal bits in both sets are the same.
 */
int
sigequalset(const sigset_t *s1, const sigset_t *s2)
{
        /*
         * We only test valid signal bits, not rubbish following MAXSIG
         * (for speed).  Algorithm:
         * if (s1 & fillset) == (s2 & fillset) then (s1 ^ s2) & fillset == 0
         */
/* see lib/libc/inc/thr_uberdata.h for why this must be true */
#if (MAXSIG > (2 * 32) && MAXSIG <= (3 * 32))
        return (!((s1->__sigbits[0] ^ s2->__sigbits[0]) |
            (s1->__sigbits[1] ^ s2->__sigbits[1]) |
```



```
            ((s1->__sigbits[2] ^ s2->__sigbits[2]) & FILLSET2)));
#else
#error "fix me: MAXSIG out of bounds"
#endif
}

/*
 * Common code for calling the user-specified signal handler.
 */
void
call_user_handler(int sig, siginfo_t *sip, ucontext_t *ucp)
{
        int i;
        ulwp_t *self = curthread;
        uberdata_t *udp = self->ul_uberdata;
        struct sigaction uact;
        volatile struct sigaction *sap;

        /*
         * If we are taking a signal while parked or about to be parked
         * on __lwp_park() then remove ourself from the sleep queue so
         * that we can grab locks.  The code in mutex_lock_queue() and
         * cond_wait_common() will detect this and deal with it when
         * __lwp_park() returns.
         */
        unsleep_self();
        set_parking_flag(self, 0);

        if (__td_event_report(self, TD_CATCHSIG, udp)) {
                self->ul_td_evbuf.eventnum = TD_CATCHSIG;
                self->ul_td_evbuf.eventdata = (void *)(intptr_t)sig;
                tdb_event(TD_CATCHSIG, udp);
        }

        /*
         * Get a self-consistent set of flags, handler, and mask
         * while holding the sig's sig_lock for the least possible time.
         * We must acquire the sig's sig_lock because some thread running
         * in sigaction() might be establishing a new signal handler.
         * The code in sigaction() acquires the writer lock; here
         * we acquire the readers lock to ehance concurrency in the
         * face of heavy signal traffic, such as generated by java.
         *
         * Locking exceptions:
         * No locking for a child of vfork().
         * If the signal is SIGPROF with an si_code of PROF_SIG,
         * then we assume that this signal was generated by
         * setitimer(ITIMER_REALPROF) set up by the dbx collector.
         * If the signal is SIGEMT with an si_code of EMT_CPCOVF,
         * then we assume that the signal was generated by
         * a hardware performance counter overflow.
         * In these cases, assume that we need no locking.  It is the
         * monitoring program's responsibility to ensure correctness.
         */
        sap = &udp->siguaction[sig].sig_uaction;
        if (self->ul_vfork ||
            (sip != NULL &&
            ((sig == SIGPROF && sip->si_code == PROF_SIG) ||
            (sig == SIGEMT && sip->si_code == EMT_CPCOVF)))) {
                /* we wish this assignment could be atomic */
                (void) memcpy(&uact, (void *)sap, sizeof (uact));
        } else {
                rwlock_t *rwlp = &udp->siguaction[sig].sig_lock;
                lrw_rdlock(rwlp);
                (void) memcpy(&uact, (void *)sap, sizeof (uact));
                if ((sig == SIGCANCEL || sig == SIGAIOCANCEL) &&
```

```
128                     (sap->sa_flags & SA_RESETHAND))
129                         sap->sa_sigaction = SIG_DFL;
130                 lrw_unlock(rwlp);
131         }

133         /*
134          * Set the proper signal mask and call the user's signal handler.
135          * (We overrode the user-requested signal mask with maskset
136          * so we currently have all blockable signals blocked.)
137          *
138          * We would like to ASSERT() that the signal is not a member of the
139          * signal mask at the previous level (ucp->uc_sigmask) or the specified
140          * signal mask for sigsuspend() or pollsys() (self->ul_tmpmask) but
141          * /proc can override this via PCSSIG, so we don't bother.
142          *
143          * We would also like to ASSERT() that the signal mask at the previous
144          * level equals self->ul_sigmask (maskset for sigsuspend() / pollsys()),
145          * but /proc can change the thread's signal mask via PCSHOLD, so we
146          * don't bother with that either.
147          */
148         ASSERT(ucp->uc_flags & UC_SIGMASK);
149         if (self->ul_sigsuspend) {
150                 ucp->uc_sigmask = self->ul_sigmask;
151                 self->ul_sigsuspend = 0;
152                 /* the sigsuspend() or pollsys() signal mask */
153                 sigorset(&uact.sa_mask, &self->ul_tmpmask);
154         } else {
155                 /* the signal mask at the previous level */
156                 sigorset(&uact.sa_mask, &ucp->uc_sigmask);
157         }
158         if (!(uact.sa_flags & SA_NODEFER))      /* add current signal */
159                 (void) sigaddset(&uact.sa_mask, sig);

161         /*
162          * Enforce the proper order for realtime signals.  Lower signals
163          * have higher priority and multiple instances of the same signal
164          * must arrive in FIFO order (NODEFER does not apply).
165          *
166          * See section 2.4.2 of POSIX.
167          */
168         if ((sig >= SIGRTMIN) && (sig <= SIGRTMAX)) {
169                 for (i = sig; i <= SIGRTMAX; i++) {
170                         (void) sigaddset(&uact.sa_mask, i);
171                 }
172         }

174         self->ul_sigmask = uact.sa_mask;
175         self->ul_siglink = ucp;
176         (void) __lwp_sigmask(SIG_SETMASK, &uact.sa_mask);

178         /*
179          * If this thread has been sent SIGCANCEL from the kernel
180          * or from pthread_cancel(), it is being asked to exit.
181          * The kernel may send SIGCANCEL without a siginfo struct.
182          * If the SIGCANCEL is process-directed (from kill() or
183          * sigqueue()), treat it as an ordinary signal.
184          */
185         if (sig == SIGCANCEL) {
186                 if (sip == NULL || SI_FROMKERNEL(sip) ||
187                     sip->si_code == SI_LWP) {
188                         do_sigcancel();
189                         goto out;
190                 }
191                 /* SIGCANCEL is ignored by default */
192                 if (uact.sa_sigaction == SIG_DFL ||
193                     uact.sa_sigaction == SIG_IGN)
```



```
194                         goto out;
195         }

197         /*
198          * If this thread has been sent SIGAIOCANCEL (SIGLWP) and
199          * we are an aio worker thread, cancel the aio request.
200          */
201         if (sig == SIGAIOCANCEL) {
202                 aio_worker_t *aiowp = pthread_getspecific(_aio_key);

204                 if (sip != NULL && sip->si_code == SI_LWP && aiowp != NULL)
205                         siglongjmp(aiowp->work_jmp_buf, 1);
206                 /* SIGLWP is ignored by default */
207                 if (uact.sa_sigaction == SIG_DFL ||
208                     uact.sa_sigaction == SIG_IGN)
209                         goto out;
210         }

212         if (!(uact.sa_flags & SA_SIGINFO))
213                 sip = NULL;
214         __sighndlr(sig, sip, ucp, uact.sa_sigaction);

216 #if defined(sparc) || defined(__sparc)
217         /*
218          * If this is a floating point exception and the queue
219          * is non-empty, pop the top entry from the queue.  This
220          * is to maintain expected behavior.
221          */
222         if (sig == SIGFPE && ucp->uc_mcontext.fpregs.fpu_qcnt) {
223                 fpregset_t *fp = &ucp->uc_mcontext.fpregs;

225                 if (--fp->fpu_qcnt > 0) {
226                         unsigned char i;
227                         struct fq *fqp;

229                         fqp = fp->fpu_q;
230                         for (i = 0; i < fp->fpu_qcnt; i++)
231                                 fqp[i] = fqp[i+1];
232                 }
233         }
234 #endif  /* sparc */

236 out:
237         (void) setcontext(ucp);
238         thr_panic("call_user_handler(): setcontext() returned");
239 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    8336 Sun Nov  2 01:27:47 2014
new/usr/src/man/man2/sigaction.2
5281 incorrect realtime signal delivery
**********************************************************
'\" te
.\" Copyright (c) 2014, Ryan Zezeski.  All Rights Reserved.
.\" Copyright (c) 2007, Sun Microsystems, Inc.  All Rights Reserved.
.\" Copyright 1989 AT&T
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH SIGACTION 2 "Oct 22, 2014"
.TH SIGACTION 2 "Mar 23, 2005"
.SH NAME
sigaction \- detailed signal management
.SH SYNOPSIS
.LP
.nf
#include <signal.h>

\fBint\fR \fBsigaction\fR(\fBint\fR \fIsig\fR, \fBconst struct sigaction *restri
     \fBstruct sigaction *restrict\fR \fIoact\fR);
.fi

.SH DESCRIPTION
.sp
.LP
The \fBsigaction()\fR function allows the calling process to examine or specify
the action to be taken on delivery of a specific signal. See
\fBsignal.h\fR(3HEAD) for an explanation of general signal concepts.
.sp
.LP
The \fIsig\fR argument specifies the signal and can be assigned any of the
signals specified in \fBsignal.h\fR(3HEAD) except  \fBSIGKILL\fR and
\fBSIGSTOP\fR.
.sp
.LP
If the argument \fIact\fR is not \fINULL\fR, it points to a structure
specifying the new action to be taken when delivering \fIsig\fR. If the
argument \fIoact\fR is not \fINULL\fR, it points to a structure where the
action previously associated with \fIsig\fR is to be stored on return from
\fBsigaction()\fR.
.sp
.LP
The \fBsigaction\fR structure includes the following members:
.sp
.in +2
.nf
void      (*sa_handler)(\|);
void      (*sa_sigaction)(int, siginfo_t *, void *);
sigset_t  sa_mask;
int       sa_flags;
.fi
.in -2

.sp
.LP
The storage occupied by \fBsa_handler\fR and \fBsa_sigaction\fR may overlap,
and a standard-conforming application (see \fBstandards\fR(5)) must not use
both simultaneously.
.sp
.LP
The \fBsa_handler\fR member identifies the action to be associated with the
specified signal, if the  \fBSA_SIGINFO\fR flag (see below) is cleared in the
\fBsa_flags\fR field of the sigaction structure. It may take any of the values
```



```
specified in \fBsignal.h\fR(3HEAD) or that of a user specified signal handler.
If the  \fBSA_SIGINFO\fR flag is set in the \fBsa_flags\fR field, the
\fBsa_sigaction\fR field specifies a signal-catching function.
.sp
.LP
The \fBsa_mask\fR member specifies a set of signals to be blocked while the
signal handler is active. On entry to the signal handler, that set of signals
is added to the set of signals already being blocked when the signal is
delivered. In addition, the signal that caused the handler to be executed will
also be blocked, unless the  \fBSA_NODEFER\fR flag has been specified.
\fBSIGSTOP\fR and  \fBSIGKILL\fR cannot be blocked (the system silently
enforces this restriction).
.sp
.LP
The \fBsa_flags\fR member specifies a set of flags used to modify the delivery
of the signal. It is formed by a logical \fBOR\fR of any of the following
values:
.sp
.ne 2
.na
\fB\fBSA_ONSTACK\fR\fR
.ad
.RS 16n
If set and the signal is caught, and if the thread that is chosen to processes
a delivered signal has an alternate signal stack declared with
\fBsigaltstack\fR(2), then it will process the signal on that stack. Otherwise,
the signal is delivered on the thread's normal stack.
.RE

.sp
.ne 2
.na
\fB\fBSA_RESETHAND\fR\fR
.ad
.RS 16n
If set and the signal is caught, the disposition of the signal is reset to
\fBSIG_DFL\fR and the signal will not be blocked on entry to the signal handler
(\fBSIGILL\fR, \fBSIGTRAP\fR, and \fBSIGPWR\fR cannot be  automatically reset
when delivered; the system silently enforces this restriction).
.RE

.sp
.ne 2
.na
\fB\fBSA_NODEFER\fR\fR
.ad
.RS 16n
If set and the signal is caught, the signal will not be automatically
blocked by the kernel while it is being caught.  Unless the signal is
a realtime signal.  Multiple instances of the same realtime signal
must be delivered in FIFO order and thus are always deferred.
If set and the signal is caught, the signal will not be automatically blocked
by the kernel while it is being caught.
.RE

.sp
.ne 2
.na
\fB\fBSA_RESTART\fR\fR
.ad
.RS 16n
If set and the signal is caught, functions that are interrupted by the
execution of this signal's handler are transparently restarted by the system,
namely \fBfcntl\fR(2), \fBioctl\fR(2), \fBwait\fR(3C), \fBwaitid\fR(2), and the
following functions on slow devices like terminals: \fBgetmsg()\fR and
\fBgetpmsg()\fR (see \fBgetmsg\fR(2));  \fBputmsg()\fR and \fBputpmsg()\fR (see
```

```
\fBputmsg\fR(2)); \fBpread()\fR, \fBread()\fR, and \fBreadv()\fR (see
\fBread\fR(2)); \fBpwrite()\fR, \fBwrite()\fR, and \fBwritev()\fR (see
\fBwrite\fR(2)); \fBrecv()\fR, \fBrecvfrom()\fR, and \fBrecvmsg()\fR (see
\fBrecv\fR(3SOCKET)); and \fBsend()\fR, \fBsendto()\fR, and \fBsendmsg()\fR
(see \fBsend\fR(3SOCKET)). Otherwise, the function returns an \fBEINTR\fR
error.
.RE

.sp
.ne 2
.na
\fB\fBSA_SIGINFO\fR\fR
.ad
.RS 16n
If cleared and the signal is caught, \fIsig\fR is passed as the only argument
to the signal-catching function. If set and the signal is caught,  two
additional arguments are passed to the signal-catching function.  If the second
argument is not equal to \fINULL\fR, it points to a \fBsiginfo_t\fR structure
containing the reason why the signal was generated (see
\fBsiginfo.h\fR(3HEAD)); the third argument points to a \fBucontext_t\fR
structure containing the receiving process's context when the signal was
delivered (see \fBucontext.h\fR(3HEAD)).
.RE

.sp
.ne 2
.na
\fB\fBSA_NOCLDWAIT\fR\fR
.ad
.RS 16n
If set and \fIsig\fR equals  \fBSIGCHLD\fR, the system will not create zombie
processes when children of the calling process exit. If the calling process
subsequently issues a \fBwait\fR(3C), it blocks until all of the calling
process's child processes terminate, and then returns \fB\(mi1\fR with
\fBerrno\fR set to \fBECHILD\fR.
.RE

.sp
.ne 2
.na
\fB\fBSA_NOCLDSTOP\fR\fR
.ad
.RS 16n
If set and \fIsig\fR equals \fBSIGCHLD\fR, \fBSIGCHLD\fR will not be sent to
the calling process when its child processes stop or continue.
.RE

.SH RETURN VALUES
.sp
.LP
Upon successful completion, \fB0\fR is returned. Otherwise, \fB\(mi1\fR is
returned, \fBerrno\fR is set to indicate the error, and no new signal handler
is installed.
.SH ERRORS
.sp
.LP
The \fBsigaction()\fR function will fail if:
.sp
.ne 2
.na
\fB\fBEINVAL\fR\fR
.ad
.RS 10n
The value of the \fIsig\fR argument is not a valid signal number or is equal to
\fBSIGKILL\fR or \fBSIGSTOP\fR. In addition, if in a multithreaded process, it
is equal to \fBSIGWAITING\fR, \fBSIGCANCEL\fR, or \fBSIGLWP\fR.
```



```
.RE

.SH ATTRIBUTES
.sp
.LP
See \fBattributes\fR(5) for descriptions of the following attributes:
.sp

.sp
.TS
box;
c | c
l | l .
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
Interface Stability     Committed
_
MT-Level        Async-Signal-Safe
_
Standard        See \fBstandards\fR(5).
.TE

.SH SEE ALSO
.sp
.LP
\fBkill\fR(1), \fBIntro\fR(2), \fBexit\fR(2), \fBfcntl\fR(2), \fBgetmsg\fR(2),
\fBioctl\fR(2), \fBkill\fR(2), \fBpause\fR(2), \fBputmsg\fR(2), \fBread\fR(2),
\fBsigaltstack\fR(2), \fBsigprocmask\fR(2), \fBsigsend\fR(2),
\fBsigsuspend\fR(2), \fBwaitid\fR(2), \fBwrite\fR(2), \fBrecv\fR(3SOCKET),
\fBsend\fR(3SOCKET), \fBsiginfo.h\fR(3HEAD), \fBsignal\fR(3C),
\fBsignal.h\fR(3HEAD), \fBsigsetops\fR(3C), \fBucontext.h\fR(3HEAD),
\fBwait\fR(3C), \fBattributes\fR(5), \fBstandards\fR(5)
.SH NOTES
.sp
.LP
The handler routine can be declared:
.sp
.in +2
.nf
void handler (int \fIsig\fR, siginfo_t *\fIsip\fR, ucontext_t *\fIucp\fR);
.fi
.in -2

.sp
.LP
The \fIsig\fR argument is the signal number. The \fIsip\fR argument is a
pointer (to space on the stack) to a \fBsiginfo_t\fR structure, which provides
additional detail about the delivery of the signal. The \fIucp\fR argument is a
pointer (again to space on the stack) to a \fBucontext_t\fR structure (defined
in <\fBsys/ucontext.h\fR>) which contains the context from before the signal.
It is not recommended that \fIucp\fR be used by the handler to restore the
context from before the signal delivery.
```

```
**********************************************************
    1606 Sun Nov  2 01:27:48 2014
new/usr/src/pkg/manifests/system-test-ostest.mf
5281 incorrect realtime signal delivery
**********************************************************
```

```
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet at
   9 # http://www.illumos.org/license/CDDL.
  10 #

  12 #
  13 # Copyright (c) 2012 by Delphix. All rights reserved.
  14 # Copyright 2014, OmniTI Computer Consulting, Inc. All rights reserved.
  15 # Copyright 2014, Ryan Zezeski. All rights reserved.
  16 #

  18 set name=pkg.fmri value=pkg:/system/test/ostest@$(PKGVERS)
  19 set name=pkg.description value="Miscellaneous OS Unit Tests"
  20 set name=pkg.summary value="OS Unit Test Suite"
  21 set name=info.classification \
  22     value=org.opensolaris.category.2008:Development/System
  23 set name=variant.arch value=$(ARCH)
  24 dir path=opt/os-tests
  25 dir path=opt/os-tests/bin
  26 dir path=opt/os-tests/runfiles
  27 dir path=opt/os-tests/tests
  28 dir path=opt/os-tests/tests/rt_delivery
  29 dir path=opt/os-tests/tests/sigqueue
  30 file path=opt/os-tests/README mode=0444
  31 file path=opt/os-tests/bin/ostest mode=0555
  32 file path=opt/os-tests/runfiles/delphix.run mode=0444
  33 file path=opt/os-tests/runfiles/omnios.run mode=0444
  34 file path=opt/os-tests/runfiles/openindiana.run mode=0444
  35 file path=opt/os-tests/tests/poll_test mode=0555
  36 file path=opt/os-tests/tests/rt_delivery/rt_delivery mode=0555
  37 file path=opt/os-tests/tests/sigqueue/sigqueue_queue_size mode=0555
  38 license cr_Sun license=cr_Sun
  39 license lic_CDDL license=lic_CDDL
  40 depend fmri=system/test/testrunner type=require
```

```
**********************************************************
     789 Sun Nov  2 01:27:48 2014
new/usr/src/test/os-tests/runfiles/delphix.run
5281 incorrect realtime signal delivery
**********************************************************
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet at
   9 # http://www.illumos.org/license/CDDL.
  10 #

  12 #
  13 # Copyright (c) 2014 by Ryan Zezeski. All rights reserved.
  14 # Copyright (c) 2012 by Delphix. All rights reserved.
  15 #

  17 [DEFAULT]
  18 pre =
  19 verbose = False
  20 quiet = False
  21 timeout = 60
  22 post =
  23 outputdir = /var/tmp/test_results

  25 [/opt/os-tests/tests/poll_test]
  26 user = root

  28 [/opt/os-tests/tests/sigqueue]
  29 tests = ['sigqueue_queue_size']

  31 [/opt/os-tests/tests/rt_delivery]
  32 tests = ['rt_delivery']
```

```
**********************************************************
     789 Sun Nov  2 01:27:48 2014
new/usr/src/test/os-tests/runfiles/omnios.run
5281 incorrect realtime signal delivery
**********************************************************
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet at
   9 # http://www.illumos.org/license/CDDL.
  10 #

  12 #
  13 # Copyright (c) 2014 by Ryan Zezeski. All rights reserved.
  14 # Copyright (c) 2012 by Delphix. All rights reserved.
  15 #

  17 [DEFAULT]
  18 pre =
  19 verbose = False
  20 quiet = False
  21 timeout = 60
  22 post =
  23 outputdir = /var/tmp/test_results

  25 [/opt/os-tests/tests/poll_test]
  26 user = root

  28 [/opt/os-tests/tests/sigqueue]
  29 tests = ['sigqueue_queue_size']

  31 [/opt/os-tests/tests/rt_delivery]
  32 tests = ['rt_delivery']
```

```
**********************************************************
     789 Sun Nov  2 01:27:48 2014
new/usr/src/test/os-tests/runfiles/openindiana.run
5281 incorrect realtime signal delivery
**********************************************************
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet at
   9 # http://www.illumos.org/license/CDDL.
  10 #

  12 #
  13 # Copyright (c) 2014 by Ryan Zezeski. All rights reserved.
  14 # Copyright (c) 2012 by Delphix. All rights reserved.
  15 #

  17 [DEFAULT]
  18 pre =
  19 verbose = False
  20 quiet = False
  21 timeout = 60
  22 post =
  23 outputdir = /var/tmp/test_results

  25 [/opt/os-tests/tests/poll_test]
  26 user = root

  28 [/opt/os-tests/tests/sigqueue]
  29 tests = ['sigqueue_queue_size']

  31 [/opt/os-tests/tests/rt_delivery]
  32 tests = ['rt_delivery']
```

```
**********************************************************
     591 Sun Nov  2 01:27:48 2014
new/usr/src/test/os-tests/tests/Makefile
5281 incorrect realtime signal delivery
**********************************************************
```

```
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet at
   9 # http://www.illumos.org/license/CDDL.
  10 #

  12 #
  13 # Copyright (c) 2014 by Ryan Zezeski. All rights reserved.
  14 # Copyright (c) 2012 by Delphix. All rights reserved.
  15 #

  17 SUBDIRS = poll sigqueue rt_delivery
  16 SUBDIRS = poll sigqueue

  19 include $(SRC)/test/Makefile.com
```

```
**********************************************************
    1043 Sun Nov  2 01:27:48 2014
new/usr/src/test/os-tests/tests/rt_delivery/Makefile
5281 incorrect realtime signal delivery
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright (c) 2014 Ryan Zezeski. All rights reserved.
#

include $(SRC)/cmd/Makefile.cmd
include $(SRC)/test/Makefile.com

PROG = rt_delivery
OBJS = $(PROG:%=%.o)
SRCS = $(OBJS:%.o=%.c)

C99MODE = -xc99=%all

ROOTOPTPKG = $(ROOT)/opt/os-tests
TESTDIR = $(ROOTOPTPKG)/tests/rt_delivery

CMDS = $(PROG:%=$(TESTDIR)/%)
$(CMDS) := FILEMODE = 0555

all: $(PROG)

$(PROG): $(OBJS)
        $(LINK.c) $(OBJS) -o $@ $(LDLIBS)
        $(POST_PROCESS)

%.o: ../%.c
        $(COMPILE.c) $<

install: all $(CMDS)

lint: lint_SRCS

clobber: clean
        -$(RM) $(PROG)

clean:
        -$(RM) $(OBJS)

$(CMDS): $(TESTDIR) $(PROG)

$(TESTDIR):
        $(INS.dir)

$(TESTDIR)/%: %
        $(INS.file)
```

```
**********************************************************
    6015 Sun Nov  2 01:27:48 2014
new/usr/src/test/os-tests/tests/rt_delivery/rt_delivery.c
5281 incorrect realtime signal delivery
**********************************************************
```

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 *
 * Copyright 2014 Ryan Zezeski.  All rights reserved.
 */
#include <signal.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

/*
 * The 'seq' struct and 'sh' handler are used to record the order in which
 * the signals are delivered to the parent process.  When a signal is
 * delivered the handler records it in seq and then increments the index.
 */
struct seq {
        int sig, num;
};

static struct seq *seqp;

/* ARGSUSED */
void
sh(int sig, siginfo_t *info, void *unused)
{
        static int i;
        seqp[i].num = info->si_value.sival_int;
        seqp[i++].sig = sig;
}

static void
test_start(const char *test_name, const char *format, ...)
{
        va_list args;

        (void) printf("TEST STARTING %s: ", test_name);

        va_start(args, format);
        (void) vprintf(format, args);
        va_end(args);
        (void) fflush(stdout);
}

static void
test_failed(const char *test_name, const char *format, ...)
{
        va_list args;

        (void) printf("TEST FAILED %s: ", test_name);

        va_start(args, format);
```



```
        (void) vprintf(format, args);
        va_end(args);

        (void) exit(-1);
}

static void
test_passed(const char *test_name)
{
        (void) printf("TEST PASS: %s\n", test_name);
        (void) fflush(stdout);
}

/*
 * Verify the delivery order of realtime (RT) signals.  RT signals were
 * introduced in POSIX.1b and differ from regular signals in several key
 * ways:
 *
 * 1. Multiple instances of the same signal are queued and must be
 * delivered in FIFO order.
 *
 * 2. If multiple different RT signals are pending then the lower numbered
 * RT signals are delivered first.  I.e., RT signals have a notion of
 * priority.
 *
 * See sections 2.4.2 (Realtime Signal Generation and Delivery) and B.2.4
 * of the POSIX.1-2008 standard for more information.
 */
static int
rt_delivery(const char *test_name, int no_defer)
{
        int i, j, sig, sival, sigend, numsig, rc, fail = 0;
        pid_t p = getpid(), chld;
        struct sigaction sa;
        sigset_t ss, ssold;
        union sigval sv;

        /*
         * Range of signals to send.  Each signal is sent 2 times to verify FIFO
         * order.
         */
        sigend = SIGRTMIN + (sysconf(_SC_SIGQUEUE_MAX) / 2);
        sigend = sigend > SIGRTMAX ? SIGRTMAX : sigend;
        numsig = (sigend - SIGRTMIN) * 2;

        seqp = malloc(sizeof (struct seq) * numsig);

        /*
         * Step 1:
         *
         * Install the signal handler (sh) on this process (the parent).
         */
        (void) sigemptyset(&ss);
        for (i = SIGRTMIN; i < sigend; i++) {
                sa.sa_sigaction = sh;
                sa.sa_flags = SA_SIGINFO;
                sa.sa_flags |= (no_defer ? SA_NODEFER : 0);
                (void) sigemptyset(&sa.sa_mask);
                sigaction(i, &sa, NULL);
                (void) sigaddset(&ss, i);
        }

        /*
         * Step 2:
         *
         * Block all signals that are about to be sent to this process.
```

```
128          * This is needed to make sure all signals are pending at once.
129          * This is related to step 4.
130          */
131         (void) sigprocmask(SIG_BLOCK, &ss, &ssold);
132
133         /*
134          * Step 3:
135          *
136          * Fork a child process to send each RT signal 2 times to the
137          * parent process.  Remember, all signals being sent are RT
138          * signals and thus are subject to explicit ordering guarantees
139          * spelled out in POSIX.1b.  Lower numbered RT signals are
140          * _always_ delivered before higher RT signals.
141          */
142         if ((chld = fork()) == 0) {
143                 for (i = SIGRTMIN; i < sigend; i++) {
144                         for (j = 0; j < 2; j++) {
145                                 sv.sival_int = j;
146                                 if (sigqueue(p, i, sv) == -1) {
147                                         perror("problem generating signal");
148                                         exit(1);
149                                 }
150                         }
151                 }
152                 exit(0);
153         }
154
155         /*
156          * Step 4:
157          *
158          * Wait for forked process to finish sending all signals.  This is
159          * to make sure that all signals are pending before allowing the
160          * kernel to deliver them to the handler.  If you don't both wait
161          * and block then the handler will process the signals as they
162          * come in and thus we will not be testing the ordering
163          * guarentees.  Related to step 2.
164          */
165         if (wait(&rc) != chld) {
166                 test_failed(test_name, "failed to wait for child\n");
167         }
168
169         if (rc != 0) {
170                 test_failed(test_name, "child exited non-zero\n");
171         }
172         (void) sigprocmask(SIG_SETMASK, &ssold, NULL);
173
174         /*
175          * Step 5:
176          *
177          * Verify that multiple pending RT signals are delivered in
178          * correct priority (lower signals first), and that multiple
179          * instances of the same RT signal are delivered in FIFO order.
180          *
181          * E.g. if SIGRTMIN is 42 then the handler should record this
182          * ordering:
183          *
184          * {42, 0}, {42, 1}, {42, 2}, {43, 0}, {43, 1}, {43, 2}, ...
185          */
186         sig = SIGRTMIN;
187         sival = 0;
188         for (i = 0; i < numsig; i++) {
189                 if ((seqp[i].sig != sig) || seqp[i].num != sival) {
190                         fail = 1;
191                 }
192                 sig = ((i + 1) % 2) == 0 ? sig + 1 : sig;
193                 sival = ((i + 1) % 2) == 0 ? 0 : sival + 1;
```



```
194         }
195
196         if (fail) {
197                 sig = SIGRTMIN;
198                 sival = 0;
199                 (void) fprintf(stderr, "Excpted Actual\n");
200                 for (i = 0; i < numsig; i++) {
201                         (void) fprintf(stderr, "{%2d,%d}  {%2d,%d}\n",
202                             sig, sival, seqp[i].sig, seqp[i].num);
203
204                         sig = ((i + 1) % 2) == 0 ? sig + 1 : sig;
205                         sival = ((i + 1) % 2) == 0 ? 0 : sival + 1;
206                 }
207                 test_failed(test_name, "incorrect order\n");
208         }
209
210         free(seqp);
211         return (0);
212 }
213
214 static void
215 rt_delivery_defer_test(void)
216 {
217         const char *test_name = __func__;
218         test_start(test_name, "verify delivery order of pending RT signals\n");
219         (void) rt_delivery(test_name, 0);
220         test_passed(test_name);
221 }
222
223 static void
224 rt_delivery_nodefer_test(void)
225 {
226         const char *test_name = __func__;
227         test_start(test_name,
228             "verify delivery order of pending RT signals w/ SA_NODEFER\n");
229         (void) rt_delivery(test_name, 1);
230         test_passed(test_name);
231 }
232
233 static void
234 run_tests(void)
235 {
236         rt_delivery_defer_test();
237         rt_delivery_nodefer_test();
238 }
239
240 /* ARGSUSED */
241 int
242 main(int argc, char *argv[])
243 {
244         run_tests();
245         return (EXIT_SUCCESS);
246 }
```