****

14352 Thu Jan 10 19:57:10 2013
new/usr/src/lib/libproc/common/Psyscall.c
3463 agent lwp clobbers amd64 abi stack redzone
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Richard Lowe <richlowe@richlowe.net>

****

```
/*
 * 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 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2013, Joyent Inc. All rights reserved.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <ctype.h>
#include <fcntl.h>
#include <string.h>
#include <memory.h>
#include <errno.h>
#include <dirent.h>
#include <limits.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <sys/stat.h>
#include <sys/resource.h>
#include <sys/param.h>
#include <sys/stack.h>
#include <sys/fault.h>
#include <sys/syscall.h>
#include <sys/sysmacros.h>

#include "libproc.h"
#include "Pcontrol.h"
#include "Putil.h"
#include "P32ton.h"
#include "Pisadep.h"

extern sigset_t blockable_sigs;
```

```
static void
Pabort_agent(struct ps_prochandle *P)
{
        int sysnum = P->status.pr_lwp.pr_syscall;
        int stop;

        dprintf("agent LWP is asleep in syscall %d\n", sysnum);
        (void) Pstop(P, 0);
        stop = Psysexit(P, sysnum, TRUE);

        if (Psetrun(P, 0, PRSABORT) == 0) {
                while (Pwait(P, 0) == -1 && errno == EINTR)
                        continue;
                (void) Psysexit(P, sysnum, stop);
                dprintf("agent LWP system call aborted\n");
        }
}
```

______***unchanged_portion_omitted_***

```
/*
 * Perform system call in controlled process.
 */
int
Psyscall(struct ps_prochandle *P,
        sysret_t *rval,         /* syscall return values */
        int sysindex,           /* system call index */
        uint_t nargs,           /* number of arguments to system call */
        argdes_t *argp)         /* argument descriptor array */
{
        int agent_created = FALSE;
        pstatus_t save_pstatus;
        argdes_t *adp;                  /* pointer to argument descriptor */
        int i;                          /* general index value */
        int model;                      /* data model */
        int error = 0;                  /* syscall errno */
        int Perr = 0;                   /* local error number */
        int sexit;                      /* old value of stop-on-syscall-exit */
        prgreg_t sp;                    /* adjusted stack pointer */
        prgreg_t ap;                    /* adjusted argument pointer */
        sigset_t unblock;

        (void) sigprocmask(SIG_BLOCK, &blockable_sigs, &unblock);

        rval->sys_rval1 = 0;            /* initialize return values */
        rval->sys_rval2 = 0;

        if (sysindex <= 0 || sysindex > PRMAXSYS || nargs > MAXARGS)
                goto bad1;      /* programming error */

        if (P->state == PS_DEAD || P->state == PS_UNDEAD || P->state == PS_IDLE)
                goto bad1;      /* dead processes can't perform system calls */

        model = P->status.pr_dmodel;
#ifndef _LP64
        /* We must be a 64-bit process to deal with a 64-bit process */
        if (model == PR_MODEL_LP64)
                goto bad9;
#endif

        /*
         * Create the /proc agent LWP in the process to do all the work.
         * (It may already exist; nested create/destroy is permitted
         * by virtue of the reference count.)
         */
        if (Pcreate_agent(P) != 0)
```

```
                goto bad8;

        /*
         * Save agent’s status to restore on exit.
         */
        agent_created = TRUE;
        save_pstatus = P->status;

        if (P->state != PS_STOP ||              /* check state of LWP */
            (P->status.pr_flags & PR_ASLEEP))
                goto bad2;

        if (Pscantext(P))                       /* bad text ? */
                goto bad3;

        /*
         * Validate arguments and compute the stack frame parameters.
         * Begin with the current stack pointer.
         */
#ifdef _LP64
        if (model == PR_MODEL_LP64) {
                sp = P->status.pr_lwp.pr_reg[R_SP] + STACK_BIAS;
#if defined(__amd64)
                /*
                 * To offset the expense of computerised subtraction, the AMD64
                 * ABI allows a process the use of a 128-byte area beyond the
                 * location pointed to by %rsp.  We must advance the agent’s
                 * stack pointer by at least the size of this region or else it
                 * may corrupt this temporary storage.
                 */
                sp -= STACK_RESERVE64;
#endif
                sp = PSTACK_ALIGN64(sp);
        } else {
#endif
                sp = (uint32_t)P->status.pr_lwp.pr_reg[R_SP];
                sp = PSTACK_ALIGN32(sp);
#ifdef _LP64
        }
#endif

        /*
         * For each AT_BYREF argument, compute the necessary
         * stack space and the object’s stack address.
         */
        for (i = 0, adp = argp; i < nargs; i++, adp++) {
                rval->sys_rval1 = i;            /* in case of error */
                switch (adp->arg_type) {
                default:                        /* programming error */
                        goto bad4;
                case AT_BYVAL:                  /* simple argument */
                        break;
                case AT_BYREF:                  /* must allocate space */
                        switch (adp->arg_inout) {
                        case AI_INPUT:
                        case AI_OUTPUT:
                        case AI_INOUT:
                                if (adp->arg_object == NULL)
                                        goto bad5;      /* programming error */
                                break;
                        default:                /* programming error */
                                goto bad6;
                        }
                        /* allocate stack space for BYREF argument */
                        if (adp->arg_size == 0 || adp->arg_size > MAXARGL)
                                goto bad7;      /* programming error */
```

```
#ifdef _LP64
                        if (model == PR_MODEL_LP64)
                                sp = PSTACK_ALIGN64(sp - adp->arg_size);
                        else
#endif
                                sp = PSTACK_ALIGN32(sp - adp->arg_size);
                        adp->arg_value = sp;    /* stack address for object */
                        break;
                }
        }
        rval->sys_rval1 = 0;                    /* in case of error */
        /*
         * Point of no return.
         * Perform the system call entry, adjusting %sp.
         * This moves the LWP to the stopped-on-syscall-entry state
         * just before the arguments to the system call are fetched.
         */
        ap = Psyscall_setup(P, nargs, sysindex, sp);
        P->flags |= SETREGS;    /* set registers before continuing */
        dprintf("Psyscall(): execute(sysindex = %d)\n", sysindex);

        /*
         * Execute the syscall instruction and stop on syscall entry.
         */
        if (execute(P, sysindex) != 0 ||
            (!Pissyscall(P, P->status.pr_lwp.pr_reg[R_PC]) &&
            !Pissyscall_prev(P, P->status.pr_lwp.pr_reg[R_PC], NULL)))
                goto bad10;

        dprintf("Psyscall(): copying arguments\n");

        /*
         * The LWP is stopped at syscall entry.
         * Copy objects to stack frame for each argument.
         */
        for (i = 0, adp = argp; i < nargs; i++, adp++) {
                rval->sys_rval1 = i;            /* in case of error */
                if (adp->arg_type != AT_BYVAL &&
                    adp->arg_inout != AI_OUTPUT) {
                        /* copy input byref parameter to process */
                        if (Pwrite(P, adp->arg_object, adp->arg_size,
                            (uintptr_t)adp->arg_value) != adp->arg_size)
                                goto bad17;
                }
        }
        rval->sys_rval1 = 0;                    /* in case of error */
        if (Psyscall_copyinargs(P, nargs, argp, ap) != 0)
                goto bad18;

        /*
         * Complete the system call.
         * This moves the LWP to the stopped-on-syscall-exit state.
         */
        dprintf("Psyscall(): set running at sysentry\n");

        sexit = Psysexit(P, sysindex, TRUE);    /* catch this syscall exit */
        do {
                if (Psetrun(P, 0, 0) == -1)
                        goto bad21;
                while (P->state == PS_RUN)
                        (void) Pwait(P, 0);
        } while (P->state == PS_STOP && P->status.pr_lwp.pr_why != PR_SYSEXIT);
        (void) Psysexit(P, sysindex, sexit);    /* restore original setting */

        /*
         * If the system call was _lwp_exit(), we expect that our last call
         * to Pwait() will yield ENOENT because the LWP no longer exists.
         */
        if (sysindex == SYS_lwp_exit && errno == ENOENT) {
                dprintf("Psyscall(): _lwp_exit successful\n");
                rval->sys_rval1 = rval->sys_rval2 = 0;
                goto out;
        }

        if (P->state != PS_STOP || P->status.pr_lwp.pr_why != PR_SYSEXIT)
                goto bad22;

        if (P->status.pr_lwp.pr_what != sysindex)
                goto bad23;

        if (!Pissyscall_prev(P, P->status.pr_lwp.pr_reg[R_PC], NULL)) {
                dprintf("Pissyscall_prev() failed\n");
                goto bad24;
        }

        dprintf("Psyscall(): caught at sysexit\n");

        /*
         * For each argument.
         */
        for (i = 0, adp = argp; i < nargs; i++, adp++) {
                rval->sys_rval1 = i;            /* in case of error */
                if (adp->arg_type != AT_BYVAL &&
                    adp->arg_inout != AI_INPUT) {
                        /* copy output byref parameter from process */
                        if (Pread(P, adp->arg_object, adp->arg_size,
                            (uintptr_t)adp->arg_value) != adp->arg_size)
                                goto bad25;
                }
        }

        if (Psyscall_copyoutargs(P, nargs, argp, ap) != 0)
                goto bad26;

        /*
         * Get the return values from the syscall.
         */
        if (P->status.pr_lwp.pr_errno) {        /* error return */
                error = P->status.pr_lwp.pr_errno;
                rval->sys_rval1 = -1L;
                rval->sys_rval2 = -1L;
                dprintf("Psyscall(%d) fails with errno %d\n",
                    sysindex, error);
        } else {                                /* normal return */
                rval->sys_rval1 = P->status.pr_lwp.pr_rval1;
                rval->sys_rval2 = P->status.pr_lwp.pr_rval2;
                dprintf("Psyscall(%d) returns 0x%lx 0x%lx\n", sysindex,
                    P->status.pr_lwp.pr_rval1, P->status.pr_lwp.pr_rval2);
        }

        goto out;

bad26:  Perr++;
bad25:  Perr++;
bad24:  Perr++;
bad23:  Perr++;
bad22:  Perr++;
bad21:  Perr++;
        Perr++;
        Perr++;
bad18:  Perr++;
bad17:  Perr++;
```

```
        Perr++;
        Perr++;
        Perr++;
        Perr++;
        Perr++;
        Perr++;
bad10:  Perr++;
bad9:   Perr++;
bad8:   Perr++;
bad7:   Perr++;
bad6:   Perr++;
bad5:   Perr++;
bad4:   Perr++;
bad3:   Perr++;
bad2:   Perr++;
bad1:   Perr++;
        error = -1;
        dprintf("Psyscall(%d) fails with local error %d\n", sysindex, Perr);

out:
        /*
         * Destroy the /proc agent LWP now (or just bump down the ref count).
         */
        if (agent_created) {
                if (P->state != PS_UNDEAD) {
                        P->status = save_pstatus;
                        P->flags |= SETREGS;
                        Psync(P);
                }
                Pdestroy_agent(P);
        }

        (void) sigprocmask(SIG_SETMASK, &unblock, NULL);
        return (error);
}
```

______***unchanged_portion_omitted_***