```
**********************************************************
    2315 Fri Jun 14 20:54:42 2019
new/usr/src/cmd/sgs/librtld_db/demo/Makefile.targ
11238 librtld_db demos should work with gcc 7
**********************************************************
#
# 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) 1995, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright 2019 OmniOS Community Edition (OmniOSce) Association.
#

$(OBJDIR)/%.o:  %.c
                $(COMPILE.c) $< -o $@
                $(POST_PROCESS_O)

$(OBJDIR)/%.o:  ../common/%.c
                $(COMPILE.c) $< -o $@
                $(POST_PROCESS_O)

# DEMO DELETE START
$(ROOTONLDBIN)/%: %
                $(INS.file)

$(ROOTONLDBIN)/$(MACH64)/%: %
                $(INS.file)
# DEMO DELETE END

$(OBJDIR)/main.o: gram.h

gram.c + gram.h: ../common/gram.y
                $(YACC) -d ../common/gram.y
                $(MV) y.tab.c gram.c
                $(MV) y.tab.h gram.h

lex.c: ../common/lex.l
                $(LEX) ../common/lex.l
                $(MV) lex.yy.c lex.c

$(PROG):        $(OBJS)
                $(LINK.c) $(OBJS) -o $@ $(LDLIBS)
# DEMO DELETE START
                $(POST_PROCESS)
# DEMO DELETE END


simp: ../tests/simp.c libsub.so.1
                $(LINK.c) $(LDFLAG) -o $@ ../tests/simp.c -R. ./libsub.so.1
```

```
libsub.so.1: ../tests/sub.c
                $(LINK.c) $(LDFLAG) -o $@ -G -fpic -hlibsub.so.1 ../tests/sub.c
                $(LINK.c) $(LDFLAG) -o $@ -G -Kpic -hlibsub.so.1 ../tests/sub.c

#
# RDB sample runs & tests
#
test-maps: simp FRC
                ./rdb -f ../tests/loadmaps ./simp

test-breaks: simp FRC
                ./rdb -f ../tests/breaks ./simp

test-steps: simp FRC
                ./rdb -f ../tests/steps ./simp

test-plt_skip: simp FRC
                ./rdb -f ../tests/plt_skip ./simp

test-sparc-regs: simp FRC
                ./rdb -f ../tests/test-sparc-regs simp

test-object-padding: simp FRC
                ./rdb -f ../tests/object_padding_maps simp

$(OBJDIR):
                -@mkdir -p $(OBJDIR)

clean: FRC
                $(RM) $(OBJS) $(CLEANFILES)

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

FRC:

# DEMO DELETE START

include         $(SRC)/cmd/sgs/Makefile.targ
# DEMO DELETE END
```

****
12209 Fri Jun 14 20:54:43 2019
new/usr/src/cmd/sgs/librtld_db/demo/common/bpt.c
11238 librtld_db demos should work with gcc 7
****
______unchanged_portion_omitted_

```
unsigned
continue_to_break(struct ps_prochandle *ph)
{
        bptlist_t       *bpt;
        pstatus_t       pstatus;
        struct iovec    piov[5];
        long            oper1, oper2, oper3, pflags = 0;
        fltset_t        faults;

        /*
         * We step by the first instruction incase their was
         * a break-point there.
         */
        (void) step_n(ph, 1, FLG_SN_NONE);

        premptyset(&faults);
        praddset(&faults, FLTBPT);
        praddset(&faults, FLTILL);
        praddset(&faults, FLTPRIV);
        praddset(&faults, FLTACCESS);
        praddset(&faults, FLTBOUNDS);
        praddset(&faults, FLTIZDIV);
        praddset(&faults, FLTSTACK);
        praddset(&faults, FLTTRACE);


        /* LINTED CONSTANT */
        while (1) {
                set_breaks(ph);
                oper1 = PCSFAULT;
                piov[0].iov_base = (caddr_t)(&oper1);
                piov[0].iov_len = sizeof (oper1);

                piov[1].iov_base = (caddr_t)(&faults);
                piov[1].iov_len = sizeof (faults);

                oper2 = PCRUN;
                piov[2].iov_base = (caddr_t)(&oper2);
                piov[2].iov_len = sizeof (oper2);
                pflags = PRCFAULT;
                piov[3].iov_base = (caddr_t)(&pflags);
                piov[3].iov_len = sizeof (pflags);

                oper3 = PCWSTOP;
                piov[4].iov_base = (caddr_t)(&oper3);
                piov[4].iov_len = sizeof (oper3);

                if (writev(ph->pp_ctlfd, piov, 5) == -1) {
                        if (errno == ENOENT) {
                                ph->pp_flags &= ~FLG_PP_PACT;

                                (void) ps_close(ph);
                                (void) printf("process terminated.\n");
                                return (0);
                        }
                        perr("ctb: PCWSTOP");
                }

                if (pread(ph->pp_statusfd, &pstatus, sizeof (pstatus), 0) == -1)
```

```
                        perr("ctb: reading status");


                if ((pstatus.pr_lwp.pr_why != PR_FAULTED) ||
                    (pstatus.pr_lwp.pr_what != FLTBPT)) {
                        const char      *fltmsg;

                        if ((pstatus.pr_lwp.pr_what <= MAXFAULT) &&
                            (pstatus.pr_lwp.pr_why == PR_FAULTED))
                                fltmsg = fault_strings[pstatus.pr_lwp.pr_what];
                        else
                                fltmsg = "<unknown error>";

                        (void) fprintf(stderr, "ctb: bad stop - stopped "
                            "on why: 0x%x what: %s(0x%x)\n",
                            pstatus.pr_lwp.pr_why, fltmsg,
                            pstatus.pr_lwp.pr_what);
                        return (0);
                }

                oper1 = PCCFAULT;
                if (writev(ph->pp_ctlfd, piov, 1) == -1)
                        perr("ctb: PCCFAULT");

                if ((bpt = find_bp(ph, pstatus.pr_lwp.pr_reg[R_PC])) ==
                    (bptlist_t *)-1) {
                        (void) fprintf(stderr,
                            "stopped at unregistered breakpoint! "
                            "addr: 0x%x\n",
                            EC_WORD(pstatus.pr_lwp.pr_reg[R_PC]));
                        break;
                }
                clear_breaks(ph);

                /*
                 * If this was a BP at which we should stop
                 */
                if (bpt->bl_flags & MASK_BP_STOP)
                        break;

                (void) step_n(ph, 1, FLG_SN_NONE);
        }

        if (bpt->bl_flags & FLG_BP_USERDEF)
                (void) printf("break point reached at addr: 0x%x\n",
                    EC_WORD(pstatus.pr_lwp.pr_reg[R_PC]));

        if (bpt->bl_flags & MASK_BP_SPECIAL)
                handle_sp_break(ph);

        if (ph->pp_flags & FLG_PP_LMAPS) {
                if (get_linkmaps(ph) != RET_OK)
                if (get_linkmaps(ph) != PS_OK)
                        (void) fprintf(stderr, "problem loading linkmaps\n");
        }

        return (bpt->bl_flags);
}
```

______unchanged_portion_omitted_

```
retc_t
step_n(struct ps_prochandle *ph, size_t count, sn_flags_e flgs)
{
        pstatus_t       pstatus;
        fltset_t        faults;
        int             i;
        long            oper;
        long            flags;
        struct iovec    piov[2];

        if (pread(ph->pp_statusfd, &pstatus, sizeof (pstatus), 0) == -1)
                perr("stn: reading status");

        piov[0].iov_base = (caddr_t)(&oper);
        piov[0].iov_len = sizeof (oper);

        premptyset(&faults);
        praddset(&faults, FLTTRACE);

        flags = PRSTEP | PRCFAULT;

        for (i = 0; i < count; i++) {
                bptlist_t       *bpt;
                uintptr_t       pc, pltbase;

                pc = pstatus.pr_lwp.pr_reg[R_PC];

                if ((bpt = find_bp(ph, pc)) != (bptlist_t *)-1) {
                        if (bpt->bl_flags & MASK_BP_SPECIAL)
                                handle_sp_break(ph);
                }

                if (flgs & FLG_SN_VERBOSE)
                        disasm(ph, 1);

                oper = PCSFAULT;
                piov[1].iov_base = (caddr_t)(&faults);
                piov[1].iov_len = sizeof (faults);

                if (writev(ph->pp_ctlfd, piov, 2) == -1)
                        perr("stn: PCSFAULT");

                oper = PCRUN;
                piov[1].iov_base = (caddr_t)(&flags);
                piov[1].iov_len = sizeof (flags);
                if (writev(ph->pp_ctlfd, piov, 2) == -1)
                        perr("stn: PCRUN(PRSETP)");

                oper = PCWSTOP;
                if (writev(ph->pp_ctlfd, piov, 1) == -1)
                        perr("stn: PCWSTOP stepping");

                if (pread(ph->pp_statusfd, &pstatus, sizeof (pstatus), 0) == -1)
                        perr("stn1: reading status");
                pc = pstatus.pr_lwp.pr_reg[R_PC];


                if ((pstatus.pr_lwp.pr_why != PR_FAULTED) ||
                    (pstatus.pr_lwp.pr_what != FLTTRACE)) {
                        (void) fprintf(stderr, "sn: bad stop - stopped on "
                            "why: 0x%x what: 0x%x\n", pstatus.pr_lwp.pr_why,
                            pstatus.pr_lwp.pr_what);
                        return (RET_FAILED);
                }

                if ((flgs & FLG_SN_PLTSKIP) &&
                    ((pltbase = is_plt(ph, pc)) != (ulong_t)0)) {
                        rd_plt_info_t   rp;
                        if (rd_plt_resolution(ph->pp_rap, pc,
                            pstatus.pr_lwp.pr_lwpid, pltbase, &rp) != RD_OK) {
                                (void) fprintf(stderr,
                                    "sn: rd_plt_resolution failed\n");
```

```
                                return (RET_FAILED);
                        }
                        if (rp.pi_skip_method == RD_RESOLVE_TARGET_STEP) {
                                unsigned        bpflags;

                                (void) set_breakpoint(ph, rp.pi_target,
                                    FLG_BP_PLTRES);
                                bpflags = continue_to_break(ph);

                                (void) delete_breakpoint(ph, rp.pi_target,
                                    FLG_BP_PLTRES);

                                if (bpflags & FLG_BP_PLTRES)
                                        (void) step_n(ph, rp.pi_nstep,
                                            FLG_SN_NONE);
                        } else if (rp.pi_skip_method == RD_RESOLVE_STEP)
                                (void) step_n(ph, rp.pi_nstep, FLG_SN_NONE);
                }
        }

        oper = PRCFAULT;
        if (writev(ph->pp_ctlfd, piov, 1) == -1)
                perr("stn: PRCFAULT");

        if ((flgs & FLG_SN_VERBOSE) && (ph->pp_flags & FLG_PP_LMAPS)) {
                if (get_linkmaps(ph) != RET_OK)
                if (get_linkmaps(ph) != PS_OK)
                        (void) fprintf(stderr, "problem loading linkmaps\n");
        }

        return (RET_OK);
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
    6890 Fri Jun 14 20:54:43 2019
new/usr/src/cmd/sgs/librtld_db/demo/common/main.c
11238 librtld_db demos should work with gcc 7
**********************************************************
______unchanged_portion_omitted_

int
main(int argc, char *argv[])
{
        int                     pctlfd;
        int                     pstatusfd;
        char                    procname[PROCSIZE];
        char                    *command;
        char                    *rdb_commands = NULL;
        pid_t                   cpid;
        pstatus_t               pstatus;
        sysset_t                sysset;
        int                     c;
        int                     error = 0;
        long                    oper;
        struct iovec            piov[2];
        extern FILE             *yyin;

        command = argv[0];

        while ((c = getopt(argc, argv, "f:")) != EOF)
                switch (c) {
                case 'f':
                        rdb_commands = optarg;
                        break;
                case '?':
                        break;
                }

        if (error || (optind == argc)) {
                (void) printf("usage: %s [-f file] executable "
                    "[executable arguments ...]\n", command);
                (void) printf("\t-f     command file\n");
                exit(1);
        }

        /*
         * set up for tracing the child.
         */
        init_proc();

        /*
         * create a child to fork and exec from.
         */
        if ((cpid = fork()) == 0) {
                (void) execv(argv[optind], &argv[optind]);
                perr(argv[optind]);
                perr(argv[1]);
        }

        if (cpid == -1) /* fork() failure */
                perr(command);

        /*
         * initialize libelf
         */
        if (elf_version(EV_CURRENT) == EV_NONE) {
                (void) fprintf(stderr, "elf_version() failed: %s\n",
                    elf_errmsg(0));
                exit(1);
```

```
        }

        /*
         * initialize librtld_db
         */
        if (rd_init(RD_VERSION) != RD_OK) {
                (void) fprintf(stderr, "librtld_db::rd_init() failed: version "
                    "submitted: %d\n", RD_VERSION);
                exit(1);
        }

        /* rd_log(1); */

        /*
         * Child should now be waiting after the successful
         * exec.
         */
        (void) snprintf(procname, PROCSIZE, "/proc/%d/ctl", EC_SWORD(cpid));
        (void) printf("parent: %d child: %d child procname: %s\n",
            EC_SWORD(getpid()), EC_SWORD(cpid), procname);
        if ((pctlfd = open(procname, O_WRONLY)) < 0) {
                perror(procname);
                (void) fprintf(stderr, "%s: can't open child %s\n",
                    command, procname);
                exit(1);
        }

        /*
         * wait for child process.
         */
        oper = PCWSTOP;
        piov[0].iov_base = (caddr_t)&oper;
        piov[0].iov_len = sizeof (oper);
        if (writev(pctlfd, piov, 1) == -1)
                perr("PCWSTOP");

        /*
         * open /proc/<cpid>/status
         */
        (void) snprintf(procname, PROCSIZE, "/proc/%d/status", EC_SWORD(cpid));
        if ((pstatusfd = open(procname, O_RDONLY)) == -1)
                perr(procname);

        if (read(pstatusfd, &pstatus, sizeof (pstatus)) == -1)
                perr("status read failed");

        /*
         * Make sure that it stopped where we expected.
         */
        while ((pstatus.pr_lwp.pr_why == PR_SYSEXIT) &&
            (pstatus.pr_lwp.pr_what == SYS_execve)) {
                long    pflags = 0;
                if (!(pstatus.pr_lwp.pr_reg[R_PS] & ERRBIT)) {
                        /* successfull exec(2) */
                        break;
                }

                oper = PCRUN;
                piov[1].iov_base = (caddr_t)&pflags;
                piov[1].iov_len = sizeof (pflags);
                if (writev(pctlfd, piov, 2) == -1)
                        perr("PCRUN1");

                oper = PCWSTOP;
                if (writev(pctlfd, piov, 1) == -1)
                        perr("PCWSTOP");
                if (read(pstatusfd, &pstatus, sizeof (pstatus)) == -1)
                        perr("status read failed");
        }

        premptyset(&sysset);
        oper = PCSEXIT;
        piov[1].iov_base = (caddr_t)&sysset;
        piov[1].iov_len = sizeof (sysset);
        if (writev(pctlfd, piov, 2) == -1)
                perr("PIOCSEXIT");

        /*
         * Did we stop where we expected ?
         */
        if ((pstatus.pr_lwp.pr_why != PR_SYSEXIT) ||
            (pstatus.pr_lwp.pr_what != SYS_execve)) {
                long    pflags = 0;

                (void) fprintf(stderr, "Didn't catch the exec, why: %d "
                    "what: %d\n", pstatus.pr_lwp.pr_why,
                    pstatus.pr_lwp.pr_what);

                oper = PCRUN;
                piov[1].iov_base = (caddr_t)&pflags;
                piov[1].iov_len = sizeof (pflags);
                if (writev(pctlfd, piov, 2) == -1)
                        perr("PCRUN2");
                exit(1);
        }

        (void) ps_init(pctlfd, pstatusfd, cpid, &proch);

        if (rdb_commands) {
                if ((yyin = fopen(rdb_commands, "r")) == NULL) {
                        (void) printf("unable to open %s for input\n",
                            rdb_commands);
                        perr("fopen");
                }
        } else {
                proch.pp_flags |= FLG_PP_PROMPT;
                rdb_prompt();
        }
        (void) yyparse();

        if (proch.pp_flags & FLG_PP_PACT) {
                long    pflags = PRCFAULT;

                (void) printf("\ncontinuing the hung process...\n");

                pctlfd = proch.pp_ctlfd;
                (void) ps_close(&proch);

                oper = PCRUN;
                piov[1].iov_base = (caddr_t)&pflags;
                piov[1].iov_len = sizeof (pflags);
                if (writev(pctlfd, piov, 2) == -1)
                        perr("PCRUN2");
                (void) close(pctlfd);
        }

        return (0);
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
    9147 Fri Jun 14 20:54:44 2019
new/usr/src/cmd/sgs/librtld_db/demo/common/ps.c
11238 librtld_db demos should work with gcc 7
**********************************************************
______unchanged_portion_omitted_

ps_err_e
ps_lgetregs(struct ps_prochandle *ph, lwpid_t lid, prgregset_t gregset)
{
        char            procname[MAXPATHLEN];
        int             lwpfd;
        lwpstatus_t     lwpstatus;

        (void) snprintf(procname, MAXPATHLEN - 1,
            "/proc/%d/lwp/%d/lwpstatus", EC_SWORD(ph->pp_pid), EC_SWORD(lid));

        if ((lwpfd = open(procname, O_RDONLY)) == -1)
                return (PS_ERR);

        if (read(lwpfd, &lwpstatus, sizeof (lwpstatus)) == -1)
                return (PS_ERR);

        memcpy(gregset, lwpstatus.pr_reg, sizeof (*gregset));
        gregset = lwpstatus.pr_reg;

        (void) close(lwpfd);
        return (PS_OK);
}
______unchanged_portion_omitted_
```