

new/usr/src/cmd/sgs/librtld\_db/demo/Makefile.targ

1

```
*****
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
*****
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 # Copyright (c) 1995, 2010, Oracle and/or its affiliates. All rights reserved.
22 # Copyright 2019 OmniOS Community Edition (OmniOSce) Association.
23 #

25 $(OBJDIR)/%.o: %.c
26     $(COMPILE.c) $< -o $@
27     $(POST_PROCESS_O)

29 $(OBJDIR)/%.o: ../common/%.c
30     $(COMPILE.c) $< -o $@
31     $(POST_PROCESS_O)

33 # DEMO DELETE START
34 $(ROOTONLDBIN)/%: %
35     $(INS.file)

37 $(ROOTONLDBIN)/$(MACH64)/%: %
38     $(INS.file)
39 # DEMO DELETE END

41 $(OBJDIR)/main.o: gram.h

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

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

52 $(PROG): $(OBJS)
53     $(LINK.c) $(OBJS) -o $@ $(LDLIBS)
54 # DEMO DELETE START
55     $(POST_PROCESS)
56 # DEMO DELETE END

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

new/usr/src/cmd/sgs/librtld\_db/demo/Makefile.targ

2

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

65 #
66 # RDB sample runs & tests
67 #
68 test-maps: simp FRC
69     ./rdb -f ../tests/loadmaps ./simp

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

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

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

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

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

86 $(OBJDIR):
87     -@mkdir -p $(OBJDIR)

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

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

95 FRC:

97 # DEMO DELETE START

99 include $(SRC)/cmd/sgs/Makefile.targ
100 # 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_____

269 unsigned
270 continue_to_break(struct ps_prochandle *ph)
271 {
272     bptlist_t     *bpt;
273     pstatus_t     pstatus;
274     struct iovec  piovs[5];
275     long          oper1, oper2, oper3, pflags = 0;
276     fltset_t      faults;

278     /*
279      * We step by the first instruction incase their was
280      * a break-point there.
281      */
282     (void) step_n(ph, 1, FLG_SN_NONE);

284     premyset(&faults);
285     praddset(&faults, FLTBPT);
286     praddset(&faults, FLTILL);
287     praddset(&faults, FLTPRIV);
288     praddset(&faults, FLTACCESS);
289     praddset(&faults, FLTBOUNDS);
290     praddset(&faults, FLTIZDIV);
291     praddset(&faults, FLTSTACK);
292     praddset(&faults, FLTTRACE);

295     /* LINTED CONSTANT */
296     while (1) {
297         set_breaks(ph);
298         oper1 = PCSFAULT;
299         piovs[0].iov_base = (caddr_t)(&oper1);
300         piovs[0].iov_len = sizeof (oper1);

302         piovs[1].iov_base = (caddr_t)(&faults);
303         piovs[1].iov_len = sizeof (faults);

305         oper2 = PCRUN;
306         piovs[2].iov_base = (caddr_t)(&oper2);
307         piovs[2].iov_len = sizeof (oper2);
308         pflags = PRCFAULT;
309         piovs[3].iov_base = (caddr_t)(&pflags);
310         piovs[3].iov_len = sizeof (pflags);

312         oper3 = PCWSTOP;
313         piovs[4].iov_base = (caddr_t)(&oper3);
314         piovs[4].iov_len = sizeof (oper3);

316         if (writev(ph->pp_ctlfd, piovs, 5) == -1) {
317             if (errno == EWOULDBLOCK) {
318                 ph->pp_flags &= ~FLG_PP_PACT;

320                 (void) ps_close(ph);
321                 (void) printf("process terminated.\n");
322                 return (0);
323             }
324             perr("ctb: PCWSTOP");
325         }

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

```

```

328         perr("ctb: reading status");

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

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

341             (void) fprintf(stderr, "ctb: bad stop - stopped "
342                             "on why: 0x%x what: %s(0x%x)\n",
343                             pstatus.pr_lwp.pr_why, fltmsg,
344                             pstatus.pr_lwp.pr_what);
345             return (0);
346         }

348         oper1 = PCCFAULT;
349         if (writev(ph->pp_ctlfd, piovs, 1) == -1)
350             perr("ctb: PCCFAULT");

352         if ((bpt = find_bp(ph, pstatus.pr_lwp.pr_reg[R_PC])) ==
353             (bptlist_t *)-1) {
354             (void) fprintf(stderr,
355                             "stopped at unregistered breakpoint! "
356                             "addr: 0x%x\n",
357                             EC_WORD(pstatus.pr_lwp.pr_reg[R_PC]));
358             break;
359         }
360         clear_breaks(ph);

362         /*
363          * If this was a BP at which we should stop
364          */
365         if (bpt->bl_flags & MASK_BP_STOP)
366             break;

368         (void) step_n(ph, 1, FLG_SN_NONE);
369     }

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

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

378     if (ph->pp_flags & FLG_PP_LMAPS) {
379         if (get_linkmaps(ph) != RET_OK)
380             if (get_linkmaps(ph) != PS_OK)
381                 (void) fprintf(stderr, "problem loading linkmaps\n");
382     }

383     return (bpt->bl_flags);
384 }
_____unchanged_portion_omitted_____

405 retc_t
406 step_n(struct ps_prochandle *ph, size_t count, sn_flags_e flgs)
407 {
408     pstatus_t     pstatus;
409     fltset_t      faults;
410     int           i;

```

```

411     long         oper;
412     long         flags;
413     struct iovec  piovp[2];

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

418     piovp[0].iov_base = (caddr_t)(&oper);
419     piovp[0].iov_len = sizeof (oper);

421     premytset(&faults);
422     praddset(&faults, FLTTRACE);

424     flags = PRSTEP | PRCFAULT;

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

430         pc = pstatus.pr_lwp.pr_reg[R_PC];

432         if ((bpt = find_bp(ph, pc)) != (bptlist_t *)-1) {
433             if (bpt->bl_flags & MASK_BP_SPECIAL)
434                 handle_sp_break(ph);
435         }

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

440         oper = PCSFAULT;
441         piovp[1].iov_base = (caddr_t)(&faults);
442         piovp[1].iov_len = sizeof (faults);

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

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

453         oper = PCWSTOP;
454         if (writev(ph->pp_ctlfd, piovp, 1) == -1)
455             perr("stn: PCWSTOP stepping");

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

462         if ((pstatus.pr_lwp.pr_why != PR_FAULTED) ||
463             (pstatus.pr_lwp.pr_what != FLTTRACE)) {
464             (void) fprintf(stderr, "sn: bad stop - stopped on "
465                 "why: 0x%x what: 0x%x\n", pstatus.pr_lwp.pr_why,
466                 pstatus.pr_lwp.pr_what);
467             return (RET_FAILED);
468         }

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

```

```

477         return (RET_FAILED);
478     }
479     if (rp.pi_skip_method == RD_RESOLVE_TARGET_STEP) {
480         unsigned         bpflags;

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

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

489         if (bpflags & FLG_BP_PLTRES)
490             (void) step_n(ph, rp.pi_nstep,
491                 FLG_SN_NONE);
492     } else if (rp.pi_skip_method == RD_RESOLVE_STEP)
493         (void) step_n(ph, rp.pi_nstep, FLG_SN_NONE);
494     }
495 }

497     oper = PRCFAULT;
498     if (writev(ph->pp_ctlfd, piovp, 1) == -1)
499         perr("stn: PRCFAULT");

501     if ((flgs & FLG_SN_VERBOSE) && (ph->pp_flags & FLG_PP_LMAPS)) {
502         if (get_linkmaps(ph) != RET_OK)
503             if (get_linkmaps(ph) != PS_OK)
504                 (void) fprintf(stderr, "problem loading linkmaps\n");
505     }

506     return (RET_OK);
507 }

```

\_\_\_\_\_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_____

127 int
128 main(int argc, char *argv[])
129 {
130     int          pctlfd;
131     int          pstatusfd;
132     char         procname[PROCSIZE];
133     char         *command;
134     char         *rdb_commands = NULL;
135     pid_t        cpid;
136     pstatus_t    pstatus;
137     sysset_t     sysset;
138     int          c;
139     int          error = 0;
140     long         oper;
141     struct iovec piovf[2];
142     extern FILE  *yyin;

144     command = argv[0];

146     while ((c = getopt(argc, argv, "f:")) != EOF)
147         switch (c) {
148             case 'f':
149                 rdb_commands = optarg;
150                 break;
151             case '?':
152                 break;
153         }

155     if (error || (optind == argc)) {
156         (void) printf("usage: %s [-f file] executable "
157             "[executable arguments ...]\n", command);
158         (void) printf("\t-f      command file\n");
159         exit(1);
160     }

162     /*
163      * set up for tracing the child.
164      */
165     init_proc();

167     /*
168      * create a child to fork and exec from.
169      */
170     if ((cpid = fork()) == 0) {
171         (void) execv(argv[optind], &argv[optind]);
172         perr(argv[optind]);
173         perr(argv[1]);
174     }

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

178     /*
179      * initialize libelf
180      */
181     if (elf_version(EV_CURRENT) == EV_NONE) {
182         (void) fprintf(stderr, "elf_version() failed: %s\n",
183             elf_errmsg(0));
184         exit(1);

```

```

185     }

187     /*
188      * initialize librtld_db
189      */
190     if (rd_init(RD_VERSION) != RD_OK) {
191         (void) fprintf(stderr, "librtld_db::rd_init() failed: version "
192             "submitted: %d\n", RD_VERSION);
193         exit(1);
194     }

196     /* rd_log(1); */

198     /*
199      * Child should now be waiting after the successful
200      * exec.
201      */
202     (void) snprintf(procname, PROCSIZE, "/proc/%d/ctl", EC_SWORD(cpid));
203     (void) printf("parent: %d child: %d child procname: %s\n",
204         EC_SWORD(getpid()), EC_SWORD(cpid), procname);
205     if ((pctlfd = open(procname, O_WRONLY)) < 0) {
206         perror(procname);
207         (void) fprintf(stderr, "%s: can't open child %s\n",
208             command, procname);
209         exit(1);
210     }

212     /*
213      * wait for child process.
214      */
215     oper = PCWSTOP;
216     piovf[0].iov_base = (caddr_t)&oper;
217     piovf[0].iov_len = sizeof (oper);
218     if (writev(pctlfd, piovf, 1) == -1)
219         perr("PCWSTOP");

221     /*
222      * open /proc/<cpid>/status
223      */
224     (void) snprintf(procname, PROCSIZE, "/proc/%d/status", EC_SWORD(cpid));
225     if ((pstatusfd = open(procname, O_RDONLY)) == -1)
226         perr(procname);

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

231     /*
232      * Make sure that it stopped where we expected.
233      */
234     while ((pstatus.pr_lwp.pr_why == PR_SYSEXIT) &&
235         (pstatus.pr_lwp.pr_what == SYS_execve)) {
236         long pflags = 0;
237         if (!(pstatus.pr_lwp.pr_reg[R_PS] & ERRLBIT)) {
238             /* successfull exec(2) */
239             break;
240         }

242         oper = PCRUN;
243         piovf[1].iov_base = (caddr_t)&pflags;
244         piovf[1].iov_len = sizeof (pflags);
245         if (writev(pctlfd, piovf, 2) == -1)
246             perr("PCRUN1");

248         oper = PCWSTOP;
249         if (writev(pctlfd, piovf, 1) == -1)
250             perr("PCWSTOP");

```

```

252         if (read(pstatusfd, &pstatus, sizeof (pstatus)) == -1)
253             perr("status read failed");
254     }

256     premyset(&sysset);
257     oper = PCSEXIT;
258     piovf[1].iov_base = (caddr_t)&sysset;
259     piovf[1].iov_len = sizeof (sysset);
260     if (writev(pctlfd, piovf, 2) == -1)
261         perr("PIOCSEXIT");

263     /*
264     * Did we stop where we expected ?
265     */
266     if ((pstatus.pr_lwp.pr_why != PR_SYSEXIT) ||
267         (pstatus.pr_lwp.pr_what != SYS_execve)) {
268         long pflags = 0;

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

274         oper = PCRUN;
275         piovf[1].iov_base = (caddr_t)&pflags;
276         piovf[1].iov_len = sizeof (pflags);
277         if (writev(pctlfd, piovf, 2) == -1)
278             perr("PCRUN2");
279         exit(1);
280     }

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

284     if (rdb_commands) {
285         if ((yyin = fopen(rdb_commands, "r")) == NULL) {
286             (void) printf("unable to open %s for input\n",
287                 rdb_commands);
288             perr("fopen");
289         }
290     } else {
291         proch.pp_flags |= FLG_PP_PROMPT;
292         rdb_prompt();
293     }
294     (void) yyparse();

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

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

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

304         oper = PCRUN;
305         piovf[1].iov_base = (caddr_t)&pflags;
306         piovf[1].iov_len = sizeof (pflags);
307         if (writev(pctlfd, piovf, 2) == -1)
308             perr("PCRUN2");
309         (void) close(pctlfd);
310     }

312     return (0);
313 }

```

unchanged portion omitted

new/usr/src/cmd/sgs/librtld\_db/demo/common/ps.c

1

\*\*\*\*\*

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

```
317 ps_err_e
318 ps_lgetregs(struct ps_prochandle *ph, lwpid_t lid, pgregset_t gregset)
319 {
320     char        procname[MAXPATHLEN];
321     int         lwpfd;
322     lwpstatus_t lwpstatus;
323
324     (void) snprintf(procname, MAXPATHLEN - 1,
325                    "/proc/%d/lwp/%d/lwpstatus", EC_SWORD(ph->pp_pid), EC_SWORD(lid));
326
327     if ((lwpfd = open(procname, O_RDONLY)) == -1)
328         return (PS_ERR);
329
330     if (read(lwpfd, &lwpstatus, sizeof (lwpstatus)) == -1)
331         return (PS_ERR);
332
333     memcpy(gregset, lwpstatus.pr_reg, sizeof (*gregset));
334     gregset = lwpstatus.pr_reg;
335
336     (void) close(lwpfd);
337     return (PS_OK);
338 }
```

unchanged portion omitted