```
**********************************************************
   28369 Thu May  3 15:22:55 2012
new/usr/src/cmd/mdb/common/mdb/mdb_main.c
2676 'mdb -f vmdump.0' ignores the -f
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Richard Lowe <richlowe@richlowe.net>
Reviewed by: Gary Mills <gary_mills@fastmail.fm>
**********************************************************
```

```
/*
 * 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 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright 2012, Josef 'Jeff' Sipek <jeffpc@31bits.net>. All rights reserved.
 */

#include <sys/types.h>
#include <sys/mman.h>
#include <sys/priocntl.h>
#include <sys/rtpriocntl.h>
#include <sys/resource.h>
#include <sys/termios.h>
#include <sys/param.h>
#include <sys/regset.h>
#include <sys/frame.h>
#include <sys/stack.h>
#include <sys/reg.h>

#include <libproc.h>
#include <libscf.h>
#include <alloca.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
#include <dlfcn.h>
#include <libctf.h>
#include <errno.h>
#include <kvm.h>

#include <mdb/mdb_lex.h>
#include <mdb/mdb_debug.h>
#include <mdb/mdb_signal.h>
#include <mdb/mdb_string.h>
#include <mdb/mdb_modapi.h>
#include <mdb/mdb_target.h>
#include <mdb/mdb_gelf.h>
#include <mdb/mdb_conf.h>
```



```
#include <mdb/mdb_err.h>
#include <mdb/mdb_io_impl.h>
#include <mdb/mdb_frame.h>
#include <mdb/mdb_set.h>
#include <kmdb/kmdb_kctl.h>
#include <mdb/mdb.h>

#ifndef STACK_BIAS
#define STACK_BIAS      0
#endif

#if defined(__sparc)
#define STACK_REGISTER  SP
#else
#define STACK_REGISTER  REG_FP
#endif

#ifdef _LP64
#define MDB_DEF_IPATH   \
        "%r/usr/platform/%p/lib/adb/%i:" \
        "%r/usr/platform/%m/lib/adb/%i:" \
        "%r/usr/lib/adb/%i"
#define MDB_DEF_LPATH   \
        "%r/usr/platform/%p/lib/mdb/%t/%i:" \
        "%r/usr/platform/%m/lib/mdb/%t/%i:" \
        "%r/usr/lib/mdb/%t/%i"
#else
#define MDB_DEF_IPATH   \
        "%r/usr/platform/%p/lib/adb:" \
        "%r/usr/platform/%m/lib/adb:" \
        "%r/usr/lib/adb"
#define MDB_DEF_LPATH   \
        "%r/usr/platform/%p/lib/mdb/%t:" \
        "%r/usr/platform/%m/lib/mdb/%t:" \
        "%r/usr/lib/mdb/%t"
#endif

#define MDB_DEF_PROMPT "> "

/*
 * Similar to the panic_* variables in the kernel, we keep some relevant
 * information stored in a set of global _mdb_abort_* variables; in the
 * event that the debugger dumps core, these will aid core dump analysis.
 */
const char *volatile _mdb_abort_str;    /* reason for failure */
siginfo_t _mdb_abort_info;              /* signal info for fatal signal */
ucontext_t _mdb_abort_ctx;              /* context fatal signal interrupted */
int _mdb_abort_rcount;                  /* number of times resume requested */
int _mdb_self_fd = -1;                  /* fd for self as for valid_frame */

static void
terminate(int status)
{
        (void) mdb_signal_blockall();
        mdb_destroy();
        exit(status);
}
______unchanged_portion_omitted_
#endif /* __x86 */

int
main(int argc, char *argv[], char *envp[])
{
        extern int mdb_kvm_is_compressed_dump(mdb_io_t *);
        mdb_tgt_ctor_f *tgt_ctor = NULL;
        const char **tgt_argv = alloca(argc * sizeof (char *));
```

```
408         int tgt_argc = 0;
409         mdb_tgt_t *tgt;
410
411         char object[MAXPATHLEN], execname[MAXPATHLEN];
412         mdb_io_t *in_io, *out_io, *err_io, *null_io;
413         struct termios tios;
414         int status, c;
415         char *p;
416
417         const char *Iflag = NULL, *Lflag = NULL, *Vflag = NULL, *pidarg = NULL;
418         int fflag = 0, Kflag = 0, Rflag = 0, Sflag = 0, Oflag = 0, Uflag = 0;
419
420         int ttylike;
421         int longmode = 0;
422
423         stack_t sigstack;
424
425         if (realpath(getexecname(), execname) == NULL) {
426                 (void) strncpy(execname, argv[0], MAXPATHLEN);
427                 execname[MAXPATHLEN - 1] = '\0';
428         }
429
430         mdb_create(execname, argv[0]);
431         bzero(tgt_argv, argc * sizeof (char *));
432         argv[0] = (char *)mdb.m_pname;
433         _mdb_self_fd = open("/proc/self/as", O_RDONLY);
434
435         mdb.m_env = envp;
436
437         out_io = mdb_fdio_create(STDOUT_FILENO);
438         mdb.m_out = mdb_iob_create(out_io, MDB_IOB_WRONLY);
439
440         err_io = mdb_fdio_create(STDERR_FILENO);
441         mdb.m_err = mdb_iob_create(err_io, MDB_IOB_WRONLY);
442         mdb_iob_clrflags(mdb.m_err, MDB_IOB_AUTOWRAP);
443
444         null_io = mdb_nullio_create();
445         mdb.m_null = mdb_iob_create(null_io, MDB_IOB_WRONLY);
446
447         in_io = mdb_fdio_create(STDIN_FILENO);
448         if ((mdb.m_termtype = getenv("TERM")) != NULL) {
449                 mdb.m_termtype = strdup(mdb.m_termtype);
450                 mdb.m_flags |= MDB_FL_TERMGUESS;
451         }
452         mdb.m_term = NULL;
453
454         mdb_dmode(mdb_dstr2mode(getenv("MDB_DEBUG")));
455         mdb.m_pgid = getpgrp();
456
457         if (getenv("_MDB_EXEC") != NULL)
458                 mdb.m_flags |= MDB_FL_EXEC;
459
460         /*
461          * Setup an alternate signal stack.  When tearing down pipelines in
462          * terminate(), we may have to destroy the stack of the context in
463          * which we are currently executing the signal handler.
464          */
465         sigstack.ss_sp = mmap(NULL, SIGSTKSZ, PROT_READ | PROT_WRITE,
466             MAP_PRIVATE | MAP_ANON, -1, 0);
467         if (sigstack.ss_sp == MAP_FAILED)
468                 die("could not allocate signal stack");
469         sigstack.ss_size = SIGSTKSZ;
470         sigstack.ss_flags = 0;
471         if (sigaltstack(&sigstack, NULL) != 0)
472                 die("could not set signal stack");
```



```
474         (void) mdb_signal_sethandler(SIGPIPE, SIG_IGN, NULL);
475         (void) mdb_signal_sethandler(SIGQUIT, SIG_IGN, NULL);
476
477         (void) mdb_signal_sethandler(SIGILL, flt_handler, NULL);
478         (void) mdb_signal_sethandler(SIGTRAP, flt_handler, NULL);
479         (void) mdb_signal_sethandler(SIGIOT, flt_handler, NULL);
480         (void) mdb_signal_sethandler(SIGEMT, flt_handler, NULL);
481         (void) mdb_signal_sethandler(SIGFPE, flt_handler, NULL);
482         (void) mdb_signal_sethandler(SIGBUS, flt_handler, NULL);
483         (void) mdb_signal_sethandler(SIGSEGV, flt_handler, NULL);
484
485         (void) mdb_signal_sethandler(SIGHUP, (mdb_signal_f *)terminate, NULL);
486         (void) mdb_signal_sethandler(SIGTERM, (mdb_signal_f *)terminate, NULL);
487
488         for (mdb.m_rdvers = RD_VERSION; mdb.m_rdvers > 0; mdb.m_rdvers--) {
489                 if (rd_init(mdb.m_rdvers) == RD_OK)
490                         break;
491         }
492
493         for (mdb.m_ctfvers = CTF_VERSION; mdb.m_ctfvers > 0; mdb.m_ctfvers--) {
494                 if (ctf_version(mdb.m_ctfvers) != -1)
495                         break;
496         }
497
498         if ((p = getenv("HISTSIZE")) != NULL && strisnum(p)) {
499                 mdb.m_histlen = strtoi(p);
500                 if (mdb.m_histlen < 1)
501                         mdb.m_histlen = 1;
502         }
503
504         while (optind < argc) {
505                 while ((c = getopt(argc, argv,
506                     "fkmo:p:s:uwyACD:FI:KL:MOP:R:SUV:W")) != (int)EOF) {
507                         switch (c) {
508                         case 'f':
509                                 fflag++;
510                                 tgt_ctor = mdb_rawfile_tgt_create;
511                                 break;
512                         case 'k':
513                                 tgt_ctor = mdb_kvm_tgt_create;
514                                 break;
515                         case 'm':
516                                 mdb.m_tgtflags |= MDB_TGT_F_NOLOAD;
517                                 mdb.m_tgtflags &= ~MDB_TGT_F_PRELOAD;
518                                 break;
519                         case 'o':
520                                 if (!mdb_set_options(optarg, TRUE))
521                                         terminate(2);
522                                 break;
523                         case 'p':
524                                 tgt_ctor = mdb_proc_tgt_create;
525                                 pidarg = optarg;
526                                 break;
527                         case 's':
528                                 if (!strisnum(optarg)) {
529                                         warn("expected integer following -s\n");
530                                         terminate(2);
531                                 }
532                                 mdb.m_symdist = (size_t)(uint_t)strtoi(optarg);
533                                 break;
534                         case 'u':
535                                 tgt_ctor = mdb_proc_tgt_create;
536                                 break;
537                         case 'w':
538                                 mdb.m_tgtflags |= MDB_TGT_F_RDWR;
539                                 break;
```

```
540                         case 'y':
541                                 mdb.m_flags |= MDB_FL_USECUP;
542                                 break;
543                         case 'A':
544                                 (void) mdb_set_options("nomods", TRUE);
545                                 break;
546                         case 'C':
547                                 (void) mdb_set_options("noctf", TRUE);
548                                 break;
549                         case 'D':
550                                 mdb_dmode(mdb_dstr2mode(optarg));
551                                 break;
552                         case 'F':
553                                 mdb.m_tgtflags |= MDB_TGT_F_FORCE;
554                                 break;
555                         case 'I':
556                                 Iflag = optarg;
557                                 break;
558                         case 'L':
559                                 Lflag = optarg;
560                                 break;
561                         case 'K':
562                                 Kflag++;
563                                 break;
564                         case 'M':
565                                 mdb.m_tgtflags |= MDB_TGT_F_PRELOAD;
566                                 mdb.m_tgtflags &= ~MDB_TGT_F_NOLOAD;
567                                 break;
568                         case 'O':
569                                 Oflag++;
570                                 break;
571                         case 'P':
572                                 if (!mdb_set_prompt(optarg))
573                                         terminate(2);
574                                 break;
575                         case 'R':
576                                 (void) strncpy(mdb.m_root, optarg, MAXPATHLEN);
577                                 mdb.m_root[MAXPATHLEN - 1] = '\0';
578                                 Rflag++;
579                                 break;
580                         case 'S':
581                                 Sflag++;
582                                 break;
583                         case 'U':
584                                 Uflag++;
585                                 break;
586                         case 'V':
587                                 Vflag = optarg;
588                                 break;
589                         case 'W':
590                                 mdb.m_tgtflags |= MDB_TGT_F_ALLOWIO;
591                                 break;
592                         case '?':
593                                 if (optopt == '?')
594                                         usage(0);
595                                 /* FALLTHROUGH */
596                         default:
597                                 usage(2);
598                         }
599                 }

601                 if (optind < argc) {
602                         const char *arg = argv[optind++];

604                         if (arg[0] == '+' && strlen(arg) == 2) {
605                                 if (arg[1] != 'o') {
```



```
606                                         warn("illegal option -- %s\n", arg);
607                                         terminate(2);
608                                 }
609                                 if (optind >= argc) {
610                                         warn("option requires an argument -- "
611                                             "%s\n", arg);
612                                         terminate(2);
613                                 }
614                                 if (!mdb_set_options(argv[optind++], FALSE))
615                                         terminate(2);
616                         } else
617                                 tgt_argv[tgt_argc++] = arg;
618                 }
619         }

621         if (rd_ctl(RD_CTL_SET_HELPPATH, (void *)mdb.m_root) != RD_OK) {
622                 warn("cannot set librtld_db helper path to %s\n", mdb.m_root);
623                 terminate(2);
624         }

626         if (mdb.m_debug & MDB_DBG_HELP)
627                 terminate(0); /* Quit here if we've printed out the tokens */


630         if (Iflag != NULL && strchr(Iflag, ';') != NULL) {
631                 warn("macro path cannot contain semicolons\n");
632                 terminate(2);
633         }

635         if (Lflag != NULL && strchr(Lflag, ';') != NULL) {
636                 warn("module path cannot contain semicolons\n");
637                 terminate(2);
638         }

640         if (Kflag || Uflag) {
641                 char *nm;

643                 if (tgt_ctor != NULL || Iflag != NULL) {
644                         warn("neither -f, -k, -p, -u, nor -I "
645                             "may be used with -K\n");
646                         usage(2);
647                 }

649                 if (Lflag != NULL)
650                         mdb_set_lpath(Lflag);

652                 if ((nm = ttyname(STDIN_FILENO)) == NULL ||
653                     strcmp(nm, "/dev/console") != 0) {
654                         /*
655                          * Due to the consequences of typing mdb -K instead of
656                          * mdb -k on a tty other than /dev/console, we require
657                          * -F when starting kmdb from a tty other than
658                          * /dev/console.
659                          */
660                         if (!(mdb.m_tgtflags & MDB_TGT_F_FORCE)) {
661                                 die("-F must also be supplied to start kmdb "
662                                     "from non-console tty\n");
663                         }

665                         if (mdb.m_termtype == NULL || (mdb.m_flags &
666                             MDB_FL_TERMGUESS)) {
667                                 if (mdb.m_termtype != NULL)
668                                         strfree(mdb.m_termtype);

670                                 if ((mdb.m_termtype = mdb_scf_console_term()) !=
671                                     NULL)
```

```
672                                     mdb.m_flags |= MDB_FL_TERMGUESS;
673                     }
674             } else {
675                     /*
676                      * When on console, $TERM (if set) takes precedence over
677                      * the SMF setting.
678                      */
679                     if (mdb.m_termtype == NULL && (mdb.m_termtype =
680                         mdb_scf_console_term()) != NULL)
681                             mdb.m_flags |= MDB_FL_TERMGUESS;
682             }

684             control_kmdb(Kflag);
685             terminate(0);
686             /*NOTREACHED*/
687     }

689     /*
690      * If standard input appears to have tty attributes, attempt to
691      * initialize a terminal i/o backend on top of stdin and stdout.
692      */
693     ttylike = (IOP_CTL(in_io, TCGETS, &tios) == 0);
694     if (ttylike) {
695             if ((mdb.m_term = mdb_termio_create(mdb.m_termtype,
696                 in_io, out_io)) == NULL) {
697                     if (!(mdb.m_flags & MDB_FL_EXEC)) {
698                             warn("term init failed: command-line editing "
699                                 "and prompt will not be available\n");
700                     }
701             } else {
702                     in_io = mdb.m_term;
703             }
704     }

706     mdb.m_in = mdb_iob_create(in_io, MDB_IOB_RDONLY);
707     if (mdb.m_term != NULL) {
708             mdb_iob_setpager(mdb.m_out, mdb.m_term);
709             if (mdb.m_flags & MDB_FL_PAGER)
710                     mdb_iob_setflags(mdb.m_out, MDB_IOB_PGENABLE);
711             else
712                     mdb_iob_clrflags(mdb.m_out, MDB_IOB_PGENABLE);
713     } else if (ttylike)
714             mdb_iob_setflags(mdb.m_in, MDB_IOB_TTYLIKE);
715     else
716             mdb_iob_setbuf(mdb.m_in, mdb_alloc(1, UM_SLEEP), 1);

718     mdb_pservice_init();
719     mdb_lex_reset();

721     if ((mdb.m_shell = getenv("SHELL")) == NULL)
722             mdb.m_shell = "/bin/sh";

724     /*
725      * If the debugger state is to be inherited from a previous instance,
726      * restore it now prior to path evaluation so that %R is updated.
727      */
728     if ((p = getenv(MDB_CONFIG_ENV_VAR)) != NULL) {
729             mdb_set_config(p);
730             (void) unsetenv(MDB_CONFIG_ENV_VAR);
731     }

733     /*
734      * Path evaluation part 1: Create the initial module path to allow
735      * the target constructor to load a support module.  Then expand
736      * any command-line arguments that modify the paths.
737      */
```



```
738     if (Iflag != NULL)
739             mdb_set_ipath(Iflag);
740     else
741             mdb_set_ipath(MDB_DEF_IPATH);

743     if (Lflag != NULL)
744             mdb_set_lpath(Lflag);
745     else
746             mdb_set_lpath(MDB_DEF_LPATH);

748     if (mdb_get_prompt() == NULL && !(mdb.m_flags & MDB_FL_ADB))
749             (void) mdb_set_prompt(MDB_DEF_PROMPT);

751     if (tgt_ctor == mdb_kvm_tgt_create) {
752             if (pidarg != NULL) {
753                     warn("-p and -k options are mutually exclusive\n");
754                     terminate(2);
755             }

757             if (tgt_argc == 0)
758                     tgt_argv[tgt_argc++] = "/dev/ksyms";
759             if (tgt_argc == 1 && strisnum(tgt_argv[0]) == 0) {
760                     if (mdb.m_tgtflags & MDB_TGT_F_ALLOWIO)
761                             tgt_argv[tgt_argc++] = "/dev/allkmem";
762                     else
763                             tgt_argv[tgt_argc++] = "/dev/kmem";
764             }
765     }

767     if (pidarg != NULL) {
768             if (tgt_argc != 0) {
769                     warn("-p may not be used with other arguments\n");
770                     terminate(2);
771             }
772             if (proc_arg_psinfo(pidarg, PR_ARG_PIDS, NULL, &status) == -1) {
773                     die("cannot attach to %s: %s\n",
774                         pidarg, Pgrab_error(status));
775             }
776             if (strchr(pidarg, '/') != NULL)
777                     (void) mdb_iob_snprintf(object, MAXPATHLEN,
778                         "%s/object/a.out", pidarg);
779             else
780                     (void) mdb_iob_snprintf(object, MAXPATHLEN,
781                         "/proc/%s/object/a.out", pidarg);
782             tgt_argv[tgt_argc++] = object;
783             tgt_argv[tgt_argc++] = pidarg;
784     }

786     /*
787      * Find the first argument that is not a special "-" token.  If one is
788      * found, we will examine this file and make some inferences below.
789      */
790     for (c = 0; c < tgt_argc && strcmp(tgt_argv[c], "-") == 0; c++)
791             continue;

793     if (c < tgt_argc) {
794             Elf32_Ehdr ehdr;
795             mdb_io_t *io;

797             /*
798              * If special "-" tokens preceded an argument, shift the entire
799              * argument list to the left to remove the leading "-" args.
800              */
801             if (c > 0) {
802                     bcopy(&tgt_argv[c], tgt_argv,
803                         sizeof (const char *) * (tgt_argc - c));
```

```
804                     tgt_argc -= c;
805             }
807             if (fflag)
808                     goto tcreate; /* skip re-exec and just create target */
810             /*
811              * If we just have an object file name, and that file doesn't
812              * exist, and it's a string of digits, infer it to be a
813              * sequence number referring to a pair of crash dump files.
814              */
815             if (tgt_argc == 1 && access(tgt_argv[0], F_OK) == -1 &&
816                 strisnum(tgt_argv[0])) {
818                     size_t len = strlen(tgt_argv[0]) + 8;
819                     const char *object = tgt_argv[0];
821                     tgt_argv[0] = mdb_alloc(len, UM_SLEEP);
822                     tgt_argv[1] = mdb_alloc(len, UM_SLEEP);
824                     (void) strcpy((char *)tgt_argv[0], "unix.");
825                     (void) strcat((char *)tgt_argv[0], object);
826                     (void) strcpy((char *)tgt_argv[1], "vmcore.");
827                     (void) strcat((char *)tgt_argv[1], object);
829                     if (access(tgt_argv[0], F_OK) == -1 &&
830                         access(tgt_argv[1], F_OK) == -1) {
831                             (void) strcpy((char *)tgt_argv[1], "vmdump.");
832                             (void) strcat((char *)tgt_argv[1], object);
833                             if (access(tgt_argv[1], F_OK) == 0) {
834                                     mdb_iob_printf(mdb.m_err,
835                                         "cannot open compressed dump; "
836                                         "decompress using savecore -f %s\n",
837                                         tgt_argv[1]);
838                                     terminate(0);
839                             }
840                     }
842                     tgt_argc = 2;
843             }
845             /*
846              * We need to open the object file in order to determine its
847              * ELF class and potentially re-exec ourself.
848              */
849             if ((io = mdb_fdio_create_path(NULL, tgt_argv[0],
850                 O_RDONLY, 0)) == NULL)
851                     die("failed to open %s", tgt_argv[0]);
853             /*
854              * Check for a single vmdump.N compressed dump file,
855              * and give a helpful message.
856              */
857             if (tgt_argc == 1) {
858                     if (mdb_kvm_is_compressed_dump(io)) {
859                             mdb_iob_printf(mdb.m_err,
860                                 "cannot open compressed dump; "
861                                 "decompress using savecore -f %s\n",
862                                 tgt_argv[0]);
863                             terminate(0);
864                     }
865             }
867             /*
868              * If the target is unknown or is not the rawfile target, do
869              * a gelf_check to determine if the file is an ELF file.  If
```



```
870              * it is not and the target is unknown, use the rawfile tgt.
871              * Otherwise an ELF-based target is needed, so we must abort.
872              */
873             if (mdb_gelf_check(io, &ehdr, ET_NONE) == -1) {
869             if (tgt_ctor != mdb_rawfile_tgt_create &&
870                 mdb_gelf_check(io, &ehdr, ET_NONE) == -1) {
874                     if (tgt_ctor != NULL) {
875                             (void) mdb_gelf_check(io, &ehdr, ET_EXEC);
876                             mdb_io_destroy(io);
877                             terminate(1);
878                     } else
879                             tgt_ctor = mdb_rawfile_tgt_create;
880             }
882             mdb_io_destroy(io);
884             if (identify_xvm_file(tgt_argv[0], &longmode) == 1) {
881             if (identify_xvm_file(tgt_argv[0], &longmode) == 1 &&
882                 !fflag) {
885 #ifdef _LP64
886                     if (!longmode)
887                             goto reexec;
888 #else
889                     if (longmode)
890                             goto reexec;
891 #endif
892                     tgt_ctor = mdb_kvm_tgt_create;
893                     goto tcreate;
894             }
894             if (tgt_ctor == mdb_rawfile_tgt_create)
895                     goto tcreate; /* skip re-exec and just create target */
896             /*
897              * The object file turned out to be a user core file (ET_CORE),
898              * and no other arguments were specified, swap 0 and 1.  The
899              * proc target will infer the executable for us.
900              */
901             if (ehdr.e_type == ET_CORE) {
902                     tgt_argv[tgt_argc++] = tgt_argv[0];
903                     tgt_argv[0] = NULL;
904                     tgt_ctor = mdb_proc_tgt_create;
905             }
907             /*
908              * If tgt_argv[1] is filled in, open it up and determine if it
909              * is a vmcore file.  If it is, gelf_check will fail and we
910              * set tgt_ctor to 'kvm'; otherwise we use the default.
911              */
912             if (tgt_argc > 1 && strcmp(tgt_argv[1], "-") != 0 &&
913                 tgt_argv[0] != NULL && pidarg == NULL) {
914                     Elf32_Ehdr chdr;
916                     if (access(tgt_argv[1], F_OK) == -1)
917                             die("failed to access %s", tgt_argv[1]);
919                     /* *.N case: drop vmdump.N from the list */
920                     if (tgt_argc == 3) {
921                             if ((io = mdb_fdio_create_path(NULL,
922                                 tgt_argv[2], O_RDONLY, 0)) == NULL)
923                                     die("failed to open %s", tgt_argv[2]);
924                             if (mdb_kvm_is_compressed_dump(io))
925                                     tgt_argv[--tgt_argc] = NULL;
926                             mdb_io_destroy(io);
927                     }
```

```
929                         if ((io = mdb_fdio_create_path(NULL, tgt_argv[1],
930                             O_RDONLY, 0)) == NULL)
931                                 die("failed to open %s", tgt_argv[1]);

933                         if (mdb_gelf_check(io, &chdr, ET_NONE) == -1)
934                                 tgt_ctor = mdb_kvm_tgt_create;

936                         mdb_io_destroy(io);
937                 }

939                 /*
940                  * At this point, we've read the ELF header for either an
941                  * object file or core into ehdr.  If the class does not match
942                  * ours, attempt to exec the mdb of the appropriate class.
943                  */
944 #ifdef _LP64
945                 if (ehdr.e_ident[EI_CLASS] == ELFCLASS32)
946                         goto reexec;
947 #else
948                 if (ehdr.e_ident[EI_CLASS] == ELFCLASS64)
949                         goto reexec;
950 #endif
951         }

953 tcreate:
954         if (tgt_ctor == NULL)
955                 tgt_ctor = mdb_proc_tgt_create;

957         tgt = mdb_tgt_create(tgt_ctor, mdb.m_tgtflags, tgt_argc, tgt_argv);

959         if (tgt == NULL) {
960                 if (errno == EINVAL)
961                         usage(2); /* target can return EINVAL to get usage */
962                 if (errno == EMDB_TGT)
963                         terminate(1); /* target already printed error msg */
964                 die("failed to initialize target");
965         }

967         mdb_tgt_activate(tgt);

969         mdb_create_loadable_disasms();

971         if (Vflag != NULL && mdb_dis_select(Vflag) == -1)
972                 warn("invalid disassembler mode -- %s\n", Vflag);


975         if (Rflag && mdb.m_term != NULL)
976                 warn("Using proto area %s\n", mdb.m_root);

978         /*
979          * If the target was successfully constructed and -O was specified,
980          * we now attempt to enter piggy-mode for debugging jurassic problems.
981          */
982         if (Oflag) {
983                 pcinfo_t pci;

985                 (void) strcpy(pci.pc_clname, "RT");

987                 if (priocntl(P_LWPID, P_MYID, PC_GETCID, (caddr_t)&pci) != -1) {
988                         pcparms_t pcp;
989                         rtparms_t *rtp = (rtparms_t *)pcp.pc_clparms;

991                         rtp->rt_pri = 35;
992                         rtp->rt_tqsecs = 0;
993                         rtp->rt_tqnsecs = RT_TQDEF;
```



```
995                         pcp.pc_cid = pci.pc_cid;

997                         if (priocntl(P_LWPID, P_MYID, PC_SETPARMS,
998                             (caddr_t)&pcp) == -1) {
999                                 warn("failed to set RT parameters");
1000                                Oflag = 0;
1001                        }
1002                } else {
1003                        warn("failed to get RT class id");
1004                        Oflag = 0;
1005                }

1007                if (mlockall(MCL_CURRENT | MCL_FUTURE) == -1) {
1008                        warn("failed to lock address space");
1009                        Oflag = 0;
1010                }

1012                if (Oflag)
1013                        mdb_printf("%s: oink, oink!\n", mdb.m_pname);
1014        }

1016        /*
1017         * Path evaluation part 2: Re-evaluate the path now that the target
1018         * is ready (and thus we have access to the real platform string).
1019         * Do this before reading ~/.mdbrc to allow path modifications prior
1020         * to performing module auto-loading.
1021         */
1022        mdb_set_ipath(mdb.m_ipathstr);
1023        mdb_set_lpath(mdb.m_lpathstr);

1025        if (!Sflag && (p = getenv("HOME")) != NULL) {
1026                char rcpath[MAXPATHLEN];
1027                mdb_io_t *rc_io;
1028                int fd;

1030                (void) mdb_iob_snprintf(rcpath, MAXPATHLEN, "%s/.mdbrc", p);
1031                fd = open64(rcpath, O_RDONLY);

1033                if (fd >= 0 && (rc_io = mdb_fdio_create_named(fd, rcpath))) {
1034                        mdb_iob_t *iob = mdb_iob_create(rc_io, MDB_IOB_RDONLY);
1035                        mdb_iob_t *old = mdb.m_in;

1037                        mdb.m_in = iob;
1038                        (void) mdb_run();
1039                        mdb.m_in = old;
1040                }
1041        }

1043        if (!(mdb.m_flags & MDB_FL_NOMODS))
1044                mdb_module_load_all(0);

1046        (void) mdb_signal_sethandler(SIGINT, int_handler, NULL);
1047        while ((status = mdb_run()) == MDB_ERR_ABORT ||
1048            status == MDB_ERR_OUTPUT) {
1049                /*
1050                 * If a write failed on stdout, give up.  A more informative
1051                 * error message will already have been printed by mdb_run().
1052                 */
1053                if (status == MDB_ERR_OUTPUT &&
1054                    mdb_iob_getflags(mdb.m_out) & MDB_IOB_ERR) {
1055                        mdb_warn("write to stdout failed, exiting\n");
1056                        break;
1057                }
1058                continue;
1059        }
```

```
1061         terminate((status == MDB_ERR_QUIT || status == 0) ? 0 : 1);
1062         /*NOTREACHED*/
1063         return (0);

1065 reexec:
1066         if ((p = strrchr(execname, '/')) == NULL)
1067                 die("cannot determine absolute pathname\n");
1068 #ifdef _LP64
1069 #ifdef __sparc
1070         (void) strcpy(p, "/../sparcv7/");
1071 #else
1072         (void) strcpy(p, "/../i86/");
1073 #endif
1074 #else
1075 #ifdef __sparc
1076         (void) strcpy(p, "/../sparcv9/");
1077 #else
1078         (void) strcpy(p, "/../amd64/");
1079 #endif
1080 #endif
1081         (void) strcat(p, mdb.m_pname);

1083         if (mdb.m_term != NULL)
1084                 (void) IOP_CTL(in_io, TCSETSW, &tios);

1086         (void) putenv("_MDB_EXEC=1");
1087         (void) execv(execname, argv);

1089         /*
1090          * If execv fails, suppress ENOEXEC.  Experience shows the most common
1091          * reason is that the machine is booted under a 32-bit kernel, in which
1092          * case it is clearer to only print the message below.
1093          */
1094         if (errno != ENOEXEC)
1095                 warn("failed to exec %s", execname);
1096 #ifdef _LP64
1097         die("64-bit %s cannot debug 32-bit program %s\n",
1098             mdb.m_pname, tgt_argv[0] ?
1099             tgt_argv[0] : tgt_argv[1]);
1100 #else
1101         die("32-bit %s cannot debug 64-bit program %s\n",
1102             mdb.m_pname, tgt_argv[0] ?
1103             tgt_argv[0] : tgt_argv[1]);
1104 #endif

1106         goto tcreate;
1107 }
```

_____***unchanged_portion_omitted_***