****

1267 Fri Nov 30 19:01:27 2012
new/usr/src/cmd/stat/Makefile
749 "/usr/bin/kstat" should be rewritten in C
Reviewed by: Garrett D'Amore <garrett@damore.org>
Reviewed by: Brendan Gregg <brendan.gregg@joyent.com>

****

```
#
# 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.
#
#ident  "%Z%%M% %I%     %E% SMI"
#
# cmd/stat/Makefile
#

include ../Makefile.cmd

SUBDIRS=        iostat mpstat vmstat fsstat kstat
SUBDIRS=        iostat mpstat vmstat fsstat

all :=          TARGET = all
install :=      TARGET = install
clean :=        TARGET = clean
clobber :=      TARGET = clobber
lint :=         TARGET = lint
_msg :=         TARGET = _msg

.KEEP_STATE:

all install lint clean clobber _msg: $(SUBDIRS)

$(SUBDIRS): FRC
        @cd $@; pwd; $(MAKE) $(MFLAGS) $(TARGET)

FRC:
```

```
**********************************************************
    1613 Fri Nov 30 19:01:28 2012
new/usr/src/cmd/stat/kstat/Makefile
749 "/usr/bin/kstat" should be rewritten in C
Reviewed by: Garrett D'Amore <garrett@damore.org>
Reviewed by: Brendan Gregg <brendan.gregg@joyent.com>
**********************************************************
```

```
#
# 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.
#

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

include $(SRC)/cmd/Makefile.cmd
include $(SRC)/cmd/stat/Makefile.stat

LDLIBS += -lavl -lcmdutils -ldevinfo -lgen -lkstat
CFLAGS += $(CCVERBOSE) -I${STATCOMMONDIR}
CERRWARN += -_gcc=-Wno-uninitialized
CERRWARN += -_gcc=-Wno-switch
CERRWARN += -_gcc=-Wno-parentheses
FILEMODE= 0555

lint := LINTFLAGS = -muxs -I$(STATCOMMONDIR)

.KEEP_STATE:

all: $(PROG)

install: all $(ROOTPROG)

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

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

clean:
        -$(RM) $(OBJS) $(COMMON_OBJS)

lint: lint_SRCS
```

```
include $(SRC)/cmd/Makefile.targ
#endif /* ! codereview */
```

```
**********************************************************
   35416 Fri Nov 30 19:01:28 2012
new/usr/src/cmd/stat/kstat/kstat.c
749 "/usr/bin/kstat" should be rewritten in C
Reviewed by: Garrett D'Amore <garrett@damore.org>
Reviewed by: Brendan Gregg <brendan.gregg@joyent.com>
**********************************************************
```

```
/*
 * 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) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2012 David Hoeppner. All rights reserved.
 */

/*
 * Display kernel statistics
 *
 * This is a reimplementation of the perl kstat command originally found
 * under usr/src/cmd/kstat/kstat.pl
 *
 * Incompatibilities:
 *      - perl regular expressions not longer supported
 *      - options checking is stricter
 *
 * Flags added:
 *      -C      similar to the -p option but value is separated by a colon
 *      -h      display help
 *      -j      json format
 */

#include <assert.h>
#include <ctype.h>
#include <errno.h>
#include <kstat.h>
#include <langinfo.h>
#include <libgen.h>
#include <limits.h>
#include <locale.h>
#include <signal.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <time.h>
#include <unistd.h>
#include <sys/list.h>
```

```
#include <sys/time.h>
#include <sys/types.h>

#include "kstat.h"
#include "statcommon.h"

char    *cmdname = "kstat";     /* Name of this command */
int     caught_cont = 0;        /* Have caught a SIGCONT */

static uint_t   g_timestamp_fmt = NODATE;

/* Helper flag - header was printed already? */
static boolean_t g_headerflg;

/* Saved command line options */
static boolean_t g_cflg = B_FALSE;
static boolean_t g_jflg = B_FALSE;
static boolean_t g_lflg = B_FALSE;
static boolean_t g_pflg = B_FALSE;
static boolean_t g_qflg = B_FALSE;
static char     *g_ks_class = "*";

/* Return zero if a selector did match */
static int      g_matched = 1;

/* Sorted list of kstat instances */
static list_t   instances_list;
static list_t   selector_list;

int
main(int argc, char **argv)
{
        ks_selector_t   *nselector;
        ks_selector_t   *uselector;
        kstat_ctl_t     *kc;
        hrtime_t        start_n;
        hrtime_t        period_n;
        boolean_t       errflg = B_FALSE;
        boolean_t       nselflg = B_FALSE;
        boolean_t       uselflg = B_FALSE;
        char            *q;
        int             count = 1;
        int             infinite_cycles = 0;
        int             interval = 0;
        int             n = 0;
        int             c, m, tmp;

        (void) setlocale(LC_ALL, "");
#if !defined(TEXT_DOMAIN)               /* Should be defined by cc -D */
#define TEXT_DOMAIN "SYS_TEST"          /* Use this only if it wasn't */
#endif
        (void) textdomain(TEXT_DOMAIN);

        /*
         * Create the selector list and a dummy default selector to match
         * everything. While we process the cmdline options we will add
         * selectors to this list.
         */
        list_create(&selector_list, sizeof (ks_selector_t),
            offsetof(ks_selector_t, ks_next));

        nselector = new_selector();

        /*
         * Parse named command line arguments.
         */
```

```c
        while ((c = getopt(argc, argv, "h?CqjlpT:m:i:n:s:c:")) != EOF)
                switch (c) {
                case 'h':
                case '?':
                        usage();
                        exit(0);
                        break;
                case 'C':
                        g_pflg = g_cflg = B_TRUE;
                        break;
                case 'q':
                        g_qflg = B_TRUE;
                        break;
                case 'j':
                        g_jflg = B_TRUE;
                        break;
                case 'l':
                        g_pflg = g_lflg = B_TRUE;
                        break;
                case 'p':
                        g_pflg = B_TRUE;
                        break;
                case 'T':
                        switch (*optarg) {
                        case 'd':
                                g_timestamp_fmt = DDATE;
                                break;
                        case 'u':
                                g_timestamp_fmt = UDATE;
                                break;
                        default:
                                errflg = B_TRUE;
                        }
                        break;
                case 'm':
                        nselflg = B_TRUE;
                        nselector->ks_module =
                            (char *)ks_safe_strdup(optarg);
                        break;
                case 'i':
                        nselflg = B_TRUE;
                        nselector->ks_instance =
                            (char *)ks_safe_strdup(optarg);
                        break;
                case 'n':
                        nselflg = B_TRUE;
                        nselector->ks_name =
                            (char *)ks_safe_strdup(optarg);
                        break;
                case 's':
                        nselflg = B_TRUE;
                        nselector->ks_statistic =
                            (char *)ks_safe_strdup(optarg);
                        break;
                case 'c':
                        g_ks_class =
                            (char *)ks_safe_strdup(optarg);
                        break;
                default:
                        errflg = B_TRUE;
                        break;
                }

        if (g_qflg && (g_jflg || g_pflg)) {
                (void) fprintf(stderr, gettext(
                    "-q and -lpj are mutually exclusive\n"));
```

```c
                errflg = B_TRUE;
        }

        if (errflg) {
                usage();
                exit(2);
        }

        argc -= optind;
        argv += optind;

        /*
         * Consume the rest of the command line. Parsing the
         * unnamed command line arguments.
         */
        while (argc--) {
                errno = 0;
                tmp = strtoul(*argv, &q, 10);
                if (tmp == ULONG_MAX && errno == ERANGE) {
                        if (n == 0) {
                                (void) fprintf(stderr, gettext(
                                    "Interval is too large\n"));
                        } else if (n == 1) {
                                (void) fprintf(stderr, gettext(
                                    "Count is too large\n"));
                        }
                        usage();
                        exit(2);
                }

                if (errno != 0 || *q != '\0') {
                        m = 0;
                        uselector = new_selector();
                        while ((q = (char *)strsep(argv, ":")) != NULL) {
                                m++;
                                if (m > 4) {
                                        free(uselector);
                                        usage();
                                        exit(2);
                                }

                                if (*q != '\0') {
                                        switch (m) {
                                        case 1:
                                                uselector->ks_module =
                                                    (char *)ks_safe_strdup(q);
                                                break;
                                        case 2:
                                                uselector->ks_instance =
                                                    (char *)ks_safe_strdup(q);
                                                break;
                                        case 3:
                                                uselector->ks_name =
                                                    (char *)ks_safe_strdup(q);
                                                break;
                                        case 4:
                                                uselector->ks_statistic =
                                                    (char *)ks_safe_strdup(q);
                                                break;
                                        default:
                                                assert(B_FALSE);
                                        }
                                }
                        }

                        if (m < 4) {
                                free(uselector);
                                usage();
                                exit(2);
                        }

                        uselflg = B_TRUE;
                        list_insert_tail(&selector_list, uselector);
                } else {
                        if (tmp < 1) {
                                if (n == 0) {
                                        (void) fprintf(stderr, gettext(
                                            "Interval must be an "
                                            "integer >= 1"));
                                } else if (n == 1) {
                                        (void) fprintf(stderr, gettext(
                                            "Count must be an integer >= 1"));
                                }
                                usage();
                                exit(2);
                        } else {
                                if (n == 0) {
                                        interval = tmp;
                                        count = -1;
                                } else if (n == 1) {
                                        count = tmp;
                                } else {
                                        usage();
                                        exit(2);
                                }
                        }
                        n++;
                }
                argv++;
        }

        /*
         * Check if we founded a named selector on the cmdline.
         */
        if (uselflg) {
                if (nselflg) {
                        (void) fprintf(stderr, gettext(
                            "module:instance:name:statistic and "
                            "-m -i -n -s are mutually exclusive"));
                        usage();
                        exit(2);
                } else {
                        free(nselector);
                }
        } else {
                list_insert_tail(&selector_list, nselector);
        }

        assert(!list_is_empty(&selector_list));

        list_create(&instances_list, sizeof (ks_instance_t),
            offsetof(ks_instance_t, ks_next));

        while ((kc = kstat_open()) == NULL) {
                if (errno == EAGAIN) {
                        (void) poll(NULL, 0, 200);
                } else {
                        perror("kstat_open");
                        exit(3);
                }
        }
```

```c
        if (count > 1) {
                if (signal(SIGCONT, cont_handler) == SIG_ERR) {
                        (void) fprintf(stderr, gettext(
                            "signal failed"));
                        exit(3);
                }
        }

        period_n = (hrtime_t)interval * NANOSEC;
        start_n = gethrtime();

        while (count == -1 || count-- > 0) {
                ks_instances_read(kc);
                ks_instances_print();

                if (interval && count) {
                        ks_sleep_until(&start_n, period_n, infinite_cycles,
                            &caught_cont);
                        (void) kstat_chain_update(kc);
                        (void) putchar('\n');
                }
        }

        (void) kstat_close(kc);

        return (g_matched);
}

/*
 * Print usage.
 */
static void
usage(void)
{
        (void) fprintf(stderr, gettext(
            "Usage:\n"
            "kstat [ -Cjlpq ] [ -T d|u ] [ -c class ]\n"
            "      [ -m module ] [ -i instance ] [ -n name ] [ -s statistic ]\n"
            "      [ interval [ count ] ]\n"
            "kstat [ -Cjlpq ] [ -T d|u ] [ -c class ]\n"
            "      [ module:instance:name:statistic ... ]\n"
            "      [ interval [ count ] ]\n"));
}

/*
 * Sort compare function.
 */
static int
compare_instances(ks_instance_t *l_arg, ks_instance_t *r_arg)
{
        int     rval;

        rval = strcasecmp(l_arg->ks_module, r_arg->ks_module);
        if (rval == 0) {
                if (l_arg->ks_instance == r_arg->ks_instance) {
                        return (strcasecmp(l_arg->ks_name, r_arg->ks_name));
                } else if (l_arg->ks_instance < r_arg->ks_instance) {
                        return (-1);
                } else {
                        return (1);
                }
        } else {
                return (rval);
        }
}
```

```
static char *
ks_safe_strdup(char *str)
{
        char    *ret;

        if (str == NULL) {
                return (NULL);
        }

        while ((ret = strdup(str)) == NULL) {
                if (errno == EAGAIN) {
                        (void) poll(NULL, 0, 200);
                } else {
                        perror("strdup");
                        exit(3);
                }
        }

        return (ret);
}

static void
ks_sleep_until(hrtime_t *wakeup, hrtime_t interval, int forever,
    int *caught_cont)
{
        hrtime_t        now, pause, pause_left;
        struct timespec pause_tv;
        int             status;

        now = gethrtime();
        pause = *wakeup + interval - now;

        if (pause <= 0 || pause < (interval / 4)) {
                if (forever || *caught_cont) {
                        *wakeup = now + interval;
                        pause = interval;
                } else {
                        pause = interval / 2;
                        *wakeup += interval;
                }
        } else {
                *wakeup += interval;
        }

        if (pause < 1000) {
                return;
        }

        pause_left = pause;
        do {
                pause_tv.tv_sec = pause_left / NANOSEC;
                pause_tv.tv_nsec = pause_left % NANOSEC;
                status = nanosleep(&pause_tv, (struct timespec *)NULL);
                if (status < 0) {
                        if (errno == EINTR) {
                                now = gethrtime();
                                pause_left = *wakeup - now;
                                if (pause_left < 1000) {
                                        return;
                                }
                        } else {
                                perror("nanosleep");
                                exit(3);
                        }
                }
        } while (status != 0);
```

```
}

/*
 * Inserts an instance in the per selector list.
 */
static void
nvpair_insert(ks_instance_t *ksi, char *name, ks_value_t *value,
    uchar_t data_type)
{
        ks_nvpair_t     *instance;
        ks_nvpair_t     *tmp;

        instance = (ks_nvpair_t *)malloc(sizeof (ks_nvpair_t));
        if (instance == NULL) {
                perror("malloc");
                exit(3);
        }

        (void) strlcpy(instance->name, name, KSTAT_STRLEN);
        (void) memcpy(&instance->value, value, sizeof (ks_value_t));
        instance->data_type = data_type;

        tmp = list_head(&ksi->ks_nvlist);
        while (tmp != NULL && strcasecmp(instance->name, tmp->name) > 0)
                tmp = list_next(&ksi->ks_nvlist, tmp);

        list_insert_before(&ksi->ks_nvlist, tmp, instance);
}

/*
 * Allocates a new all-matching selector.
 */
static ks_selector_t *
new_selector(void)
{
        ks_selector_t   *selector;

        selector = (ks_selector_t *)malloc(sizeof (ks_selector_t));
        if (selector == NULL) {
                perror("malloc");
                exit(3);
        }

        list_link_init(&selector->ks_next);

        selector->ks_module = "*";
        selector->ks_instance = "*";
        selector->ks_name = "*";
        selector->ks_statistic = "*";

        return (selector);
}

/*
 * This function was taken from the perl kstat module code - please
 * see for further comments there.
 */
static kstat_raw_reader_t
lookup_raw_kstat_fn(char *module, char *name)
{
        char            key[KSTAT_STRLEN * 2];
        register char   *f, *t;
        int             n = 0;

        for (f = module, t = key; *f != '\0'; f++, t++) {
                while (*f != '\0' && isdigit(*f))
```

```
                        f++;
                *t = *f;
        }
        *t++ = ':';

        for (f = name; *f != '\0'; f++, t++) {
                while (*f != '\0' && isdigit(*f))
                        f++;
                *t = *f;
        }
        *t = '\0';

        while (ks_raw_lookup[n].fn != NULL) {
                if (strncmp(ks_raw_lookup[n].name, key, strlen(key)) == 0)
                        return (ks_raw_lookup[n].fn);
                n++;
        }

        return (0);
}

/*
 * Iterate over all kernel statistics and save matches.
 */
static void
ks_instances_read(kstat_ctl_t *kc)
{
        kstat_raw_reader_t save_raw = NULL;
        kid_t           id;
        ks_selector_t   *selector;
        ks_instance_t   *ksi;
        ks_instance_t   *tmp;
        kstat_t         *kp;
        boolean_t       skip;
        char            *ks_number;

        for (kp = kc->kc_chain; kp != NULL; kp = kp->ks_next) {
                /* Don't bother storing the kstat headers */
                if (strncmp(kp->ks_name, "kstat_", 6) == 0) {
                        continue;
                }

                /* Don't bother storing raw stats we don't understand */
                if (kp->ks_type == KSTAT_TYPE_RAW) {
                        save_raw = lookup_raw_kstat_fn(kp->ks_module,
                            kp->ks_name);
                        if (save_raw == NULL) {
#ifdef REPORT_UNKNOWN
                                (void) fprintf(stderr,
                                    "Unknown kstat type %s:%d:%s - "
                                    "%d of size %d\n", kp->ks_module,
                                    kp->ks_instance, kp->ks_name,
                                    kp->ks_ndata, kp->ks_data_size);
#endif
                                continue;
                        }
                }

                /*
                 * Iterate over the list of selectors and skip
                 * instances we dont want. We filter for statistics
                 * later, as we dont know them yet.
                 */
                skip = B_FALSE;
                (void) asprintf(&ks_number, "%d", kp->ks_instance);
                selector = list_head(&selector_list);
```

```
                while (selector != NULL) {
                        if (!(gmatch(kp->ks_module, selector->ks_module) != 0 &&
                            gmatch(ks_number, selector->ks_instance) != 0 &&
                            gmatch(kp->ks_name, selector->ks_name) != 0 &&
                            gmatch(kp->ks_class, g_ks_class))) {
                                skip = B_TRUE;
                        }
                        selector = list_next(&selector_list, selector);
                }

                free(ks_number);

                if (skip) {
                        continue;
                }

                /*
                 * Allocate a new instance and fill in the values
                 * we know so far.
                 */
                ksi = (ks_instance_t *)malloc(sizeof (ks_instance_t));
                if (ksi == NULL) {
                        perror("malloc");
                        exit(3);
                }

                list_link_init(&ksi->ks_next);

                (void) strlcpy(ksi->ks_module, kp->ks_module, KSTAT_STRLEN);
                (void) strlcpy(ksi->ks_name, kp->ks_name, KSTAT_STRLEN);
                (void) strlcpy(ksi->ks_class, kp->ks_class, KSTAT_STRLEN);

                ksi->ks_instance = kp->ks_instance;
                ksi->ks_snaptime = kp->ks_snaptime;
                ksi->ks_type = kp->ks_type;

                list_create(&ksi->ks_nvlist, sizeof (ks_nvpair_t),
                    offsetof(ks_nvpair_t, nv_next));

                SAVE_HRTIME_X(ksi, "crtime", kp->ks_crtime);
                SAVE_HRTIME_X(ksi, "snaptime", kp->ks_snaptime);
                if (g_pflg) {
                        SAVE_STRING_X(ksi, "class", kp->ks_class);
                }

                /* Insert this instance into a sorted list */
                tmp = list_head(&instances_list);
                while (tmp != NULL && compare_instances(ksi, tmp) > 0)
                        tmp = list_next(&instances_list, tmp);

                list_insert_before(&instances_list, tmp, ksi);

                /* Read the actual statistics */
                id = kstat_read(kc, kp, NULL);
                if (id == -1) {
#ifdef REPORT_UNKNOWN
                        perror("kstat_read");
#endif
                        continue;
                }

                switch (kp->ks_type) {
                case KSTAT_TYPE_RAW:
                        save_raw(kp, ksi);
                        break;
                case KSTAT_TYPE_NAMED:
```

```c
			save_named(kp, ksi);
			break;
		case KSTAT_TYPE_INTR:
			save_intr(kp, ksi);
			break;
		case KSTAT_TYPE_IO:
			save_io(kp, ksi);
			break;
		case KSTAT_TYPE_TIMER:
			save_timer(kp, ksi);
			break;
		default:
			assert(B_FALSE); /* Invalid type */
			break;
		}
	}
}

/*
 * Print the value of a name-value pair.
 */
static void
ks_value_print(ks_nvpair_t *nvpair)
{
	switch (nvpair->data_type) {
	case KSTAT_DATA_CHAR:
		(void) fprintf(stdout, "%s", nvpair->value.c);
		break;
	case KSTAT_DATA_INT32:
		(void) fprintf(stdout, "%d", nvpair->value.i32);
		break;
	case KSTAT_DATA_UINT32:
		(void) fprintf(stdout, "%u", nvpair->value.ui32);
		break;
	case KSTAT_DATA_INT64:
		(void) fprintf(stdout, "%lld", nvpair->value.i64);
		break;
	case KSTAT_DATA_UINT64:
		(void) fprintf(stdout, "%llu", nvpair->value.ui64);
		break;
	case KSTAT_DATA_STRING:
		(void) fprintf(stdout, "%s", KSTAT_NAMED_STR_PTR(nvpair));
		break;
	case KSTAT_DATA_HRTIME:
		if (nvpair->value.ui64 == 0)
			(void) fprintf(stdout, "0");
		else
			(void) fprintf(stdout, "%.9f",
			    nvpair->value.ui64 / 1000000000.0);
		break;
	default:
		assert(B_FALSE);
	}
}

/*
 * Print a single instance.
 */
static void
ks_instance_print(ks_instance_t *ksi, ks_nvpair_t *nvpair)
{
	if (g_headerflg) {
		if (!g_pflg) {
			(void) fprintf(stdout, DFLT_FMT,
			    ksi->ks_module, ksi->ks_instance,
			    ksi->ks_name, ksi->ks_class);
```

```c
		}
		g_headerflg = B_FALSE;
	}

	if (g_pflg) {
		(void) fprintf(stdout, KS_PFMT,
		    ksi->ks_module, ksi->ks_instance,
		    ksi->ks_name, nvpair->name);
		if (!g_lflg) {
			(void) putchar(g_cflg ? ':': '\t');
			ks_value_print(nvpair);
		}
	} else {
		(void) fprintf(stdout, KS_DFMT, nvpair->name);
		ks_value_print(nvpair);
	}

	(void) putchar('\n');
}

/*
 * Print a single instance in JSON format.
 */
static void
ks_instance_print_json(ks_instance_t *ksi, ks_nvpair_t *nvpair)
{
	if (g_headerflg) {
		(void) fprintf(stdout, JSON_FMT,
		    ksi->ks_module, ksi->ks_instance,
		    ksi->ks_name, ksi->ks_class,
		    ksi->ks_type);

		if (ksi->ks_snaptime == 0)
			(void) fprintf(stdout, "\t\"snaptime\": 0,\n");
		else
			(void) fprintf(stdout, "\t\"snaptime\": %.9f,\n",
			    ksi->ks_snaptime / 1000000000.0);

		(void) fprintf(stdout, "\t\"data\": {\n");

		g_headerflg = B_FALSE;
	}

	(void) fprintf(stdout, KS_JFMT, nvpair->name);
	if (nvpair->data_type == KSTAT_DATA_STRING) {
		(void) putchar('\"');
		ks_value_print(nvpair);
		(void) putchar('\"');
	} else {
		ks_value_print(nvpair);
	}
	if (nvpair != list_tail(&ksi->ks_nvlist))
		(void) putchar(',');

	(void) putchar('\n');
}

/*
 * Print all instances.
 */
static void
ks_instances_print(void)
{
	ks_selector_t	*selector;
	ks_instance_t	*ksi, *ktmp;
	ks_nvpair_t	*nvpair, *ntmp;
```

```
        void            (*ks_print_fn)(ks_instance_t *, ks_nvpair_t *);

        if (g_timestamp_fmt != NODATE)
                print_timestamp(g_timestamp_fmt);

        if (g_jflg) {
                ks_print_fn = &ks_instance_print_json;
                (void) putchar('[');
        } else {
                ks_print_fn = &ks_instance_print;
        }

        /* Iterate over each selector */
        selector = list_head(&selector_list);
        while (selector != NULL) {

                /* Iterate over each instance */
                for (ksi = list_head(&instances_list); ksi != NULL;
                    ksi = list_next(&instances_list, ksi)) {

                        /* Finally iterate over each statistic */
                        g_headerflg = B_TRUE;
                        for (nvpair = list_head(&ksi->ks_nvlist);
                            nvpair != NULL;
                            nvpair = list_next(&ksi->ks_nvlist, nvpair)) {
                                if (gmatch(nvpair->name,
                                    selector->ks_statistic) == 0)
                                        continue;

                                g_matched = 0;
                                if (!g_qflg)
                                        (*ks_print_fn)(ksi, nvpair);
                        }

                        if (!g_headerflg) {
                                if (g_jflg) {
                                        (void) fprintf(stdout, "\t}\n}");
                                        if (ksi != list_tail(&instances_list))
                                                (void) putchar(',');
                                } else if (!g_pflg) {
                                        (void) putchar('\n');
                                }
                        }
                }

                selector = list_next(&selector_list, selector);
        }

        if (g_jflg)
                (void) fprintf(stdout, "]\n");

        (void) fflush(stdout);

        /* Free the instances list */
        ksi = list_head(&instances_list);
        while (ksi != NULL) {
                nvpair = list_head(&ksi->ks_nvlist);
                while (nvpair != NULL) {
                        ntmp = nvpair;
                        nvpair = list_next(&ksi->ks_nvlist, nvpair);
                        list_remove(&ksi->ks_nvlist, ntmp);
                        if (ntmp->data_type == KSTAT_DATA_STRING)
                                free(ntmp->value.str.addr.ptr);
                        free(ntmp);
                }
```

```
                ktmp = ksi;
                ksi = list_next(&instances_list, ksi);
                list_remove(&instances_list, ktmp);
                list_destroy(&ktmp->ks_nvlist);
                free(ktmp);
        }
}

static void
save_cpu_stat(kstat_t *kp, ks_instance_t *ksi)
{
        cpu_stat_t      *stat;
        cpu_sysinfo_t   *sysinfo;
        cpu_syswait_t   *syswait;
        cpu_vminfo_t    *vminfo;

        stat = (cpu_stat_t *)(kp->ks_data);
        sysinfo = &stat->cpu_sysinfo;
        syswait = &stat->cpu_syswait;
        vminfo  = &stat->cpu_vminfo;

        SAVE_UINT32_X(ksi, "idle", sysinfo->cpu[CPU_IDLE]);
        SAVE_UINT32_X(ksi, "user", sysinfo->cpu[CPU_USER]);
        SAVE_UINT32_X(ksi, "kernel", sysinfo->cpu[CPU_KERNEL]);
        SAVE_UINT32_X(ksi, "wait", sysinfo->cpu[CPU_WAIT]);
        SAVE_UINT32_X(ksi, "wait_io", sysinfo->cpu[W_IO]);
        SAVE_UINT32_X(ksi, "wait_swap", sysinfo->cpu[W_SWAP]);
        SAVE_UINT32_X(ksi, "wait_pio", sysinfo->cpu[W_PIO]);
        SAVE_UINT32(ksi, sysinfo, bread);
        SAVE_UINT32(ksi, sysinfo, bwrite);
        SAVE_UINT32(ksi, sysinfo, lread);
        SAVE_UINT32(ksi, sysinfo, lwrite);
        SAVE_UINT32(ksi, sysinfo, phread);
        SAVE_UINT32(ksi, sysinfo, phwrite);
        SAVE_UINT32(ksi, sysinfo, pswitch);
        SAVE_UINT32(ksi, sysinfo, trap);
        SAVE_UINT32(ksi, sysinfo, intr);
        SAVE_UINT32(ksi, sysinfo, syscall);
        SAVE_UINT32(ksi, sysinfo, sysread);
        SAVE_UINT32(ksi, sysinfo, syswrite);
        SAVE_UINT32(ksi, sysinfo, sysfork);
        SAVE_UINT32(ksi, sysinfo, sysvfork);
        SAVE_UINT32(ksi, sysinfo, sysexec);
        SAVE_UINT32(ksi, sysinfo, readch);
        SAVE_UINT32(ksi, sysinfo, writech);
        SAVE_UINT32(ksi, sysinfo, rcvint);
        SAVE_UINT32(ksi, sysinfo, xmtint);
        SAVE_UINT32(ksi, sysinfo, mdmint);
        SAVE_UINT32(ksi, sysinfo, rawch);
        SAVE_UINT32(ksi, sysinfo, canch);
        SAVE_UINT32(ksi, sysinfo, outch);
        SAVE_UINT32(ksi, sysinfo, msg);
        SAVE_UINT32(ksi, sysinfo, sema);
        SAVE_UINT32(ksi, sysinfo, namei);
        SAVE_UINT32(ksi, sysinfo, ufsiget);
        SAVE_UINT32(ksi, sysinfo, ufsdirblk);
        SAVE_UINT32(ksi, sysinfo, ufsipage);
        SAVE_UINT32(ksi, sysinfo, ufsinopage);
        SAVE_UINT32(ksi, sysinfo, inodeovf);
        SAVE_UINT32(ksi, sysinfo, fileovf);
        SAVE_UINT32(ksi, sysinfo, procovf);
        SAVE_UINT32(ksi, sysinfo, intrthread);
        SAVE_UINT32(ksi, sysinfo, intrblk);
        SAVE_UINT32(ksi, sysinfo, idlethread);
        SAVE_UINT32(ksi, sysinfo, inv_swtch);
        SAVE_UINT32(ksi, sysinfo, nthreads);
```

```c
        SAVE_UINT32(ksi, sysinfo, cpumigrate);
        SAVE_UINT32(ksi, sysinfo, xcalls);
        SAVE_UINT32(ksi, sysinfo, mutex_adenters);
        SAVE_UINT32(ksi, sysinfo, rw_rdfails);
        SAVE_UINT32(ksi, sysinfo, rw_wrfails);
        SAVE_UINT32(ksi, sysinfo, modload);
        SAVE_UINT32(ksi, sysinfo, modunload);
        SAVE_UINT32(ksi, sysinfo, bawrite);
#ifdef  STATISTICS      /* see header file */
        SAVE_UINT32(ksi, sysinfo, rw_enters);
        SAVE_UINT32(ksi, sysinfo, win_uo_cnt);
        SAVE_UINT32(ksi, sysinfo, win_uu_cnt);
        SAVE_UINT32(ksi, sysinfo, win_so_cnt);
        SAVE_UINT32(ksi, sysinfo, win_su_cnt);
        SAVE_UINT32(ksi, sysinfo, win_suo_cnt);
#endif

        SAVE_INT32(ksi, syswait, iowait);
        SAVE_INT32(ksi, syswait, swap);
        SAVE_INT32(ksi, syswait, physio);

        SAVE_UINT32(ksi, vminfo, pgrec);
        SAVE_UINT32(ksi, vminfo, pgfrec);
        SAVE_UINT32(ksi, vminfo, pgin);
        SAVE_UINT32(ksi, vminfo, pgpgin);
        SAVE_UINT32(ksi, vminfo, pgout);
        SAVE_UINT32(ksi, vminfo, pgpgout);
        SAVE_UINT32(ksi, vminfo, swapin);
        SAVE_UINT32(ksi, vminfo, pgswapin);
        SAVE_UINT32(ksi, vminfo, swapout);
        SAVE_UINT32(ksi, vminfo, pgswapout);
        SAVE_UINT32(ksi, vminfo, zfod);
        SAVE_UINT32(ksi, vminfo, dfree);
        SAVE_UINT32(ksi, vminfo, scan);
        SAVE_UINT32(ksi, vminfo, rev);
        SAVE_UINT32(ksi, vminfo, hat_fault);
        SAVE_UINT32(ksi, vminfo, as_fault);
        SAVE_UINT32(ksi, vminfo, maj_fault);
        SAVE_UINT32(ksi, vminfo, cow_fault);
        SAVE_UINT32(ksi, vminfo, prot_fault);
        SAVE_UINT32(ksi, vminfo, softlock);
        SAVE_UINT32(ksi, vminfo, kernel_asflt);
        SAVE_UINT32(ksi, vminfo, pgrrun);
        SAVE_UINT32(ksi, vminfo, execpgin);
        SAVE_UINT32(ksi, vminfo, execpgout);
        SAVE_UINT32(ksi, vminfo, execfree);
        SAVE_UINT32(ksi, vminfo, anonpgin);
        SAVE_UINT32(ksi, vminfo, anonpgout);
        SAVE_UINT32(ksi, vminfo, anonfree);
        SAVE_UINT32(ksi, vminfo, fspgin);
        SAVE_UINT32(ksi, vminfo, fspgout);
        SAVE_UINT32(ksi, vminfo, fsfree);
}

static void
save_var(kstat_t *kp, ks_instance_t *ksi)
{
        struct var      *var = (struct var *)(kp->ks_data);

        assert(kp->ks_data_size == sizeof (struct var));

        SAVE_INT32(ksi, var, v_buf);
        SAVE_INT32(ksi, var, v_call);
        SAVE_INT32(ksi, var, v_proc);
        SAVE_INT32(ksi, var, v_maxupttl);
        SAVE_INT32(ksi, var, v_nglobpris);
```

```c
        SAVE_INT32(ksi, var, v_maxsyspri);
        SAVE_INT32(ksi, var, v_clist);
        SAVE_INT32(ksi, var, v_maxup);
        SAVE_INT32(ksi, var, v_hbuf);
        SAVE_INT32(ksi, var, v_hmask);
        SAVE_INT32(ksi, var, v_pbuf);
        SAVE_INT32(ksi, var, v_sptmap);
        SAVE_INT32(ksi, var, v_maxpmem);
        SAVE_INT32(ksi, var, v_autoup);
        SAVE_INT32(ksi, var, v_bufhwm);
}

static void
save_ncstats(kstat_t *kp, ks_instance_t *ksi)
{
        struct ncstats  *ncstats = (struct ncstats *)(kp->ks_data);

        assert(kp->ks_data_size == sizeof (struct ncstats));

        SAVE_INT32(ksi, ncstats, hits);
        SAVE_INT32(ksi, ncstats, misses);
        SAVE_INT32(ksi, ncstats, enters);
        SAVE_INT32(ksi, ncstats, dbl_enters);
        SAVE_INT32(ksi, ncstats, long_enter);
        SAVE_INT32(ksi, ncstats, long_look);
        SAVE_INT32(ksi, ncstats, move_to_front);
        SAVE_INT32(ksi, ncstats, purges);
}

static void
save_sysinfo(kstat_t *kp, ks_instance_t *ksi)
{
        sysinfo_t       *sysinfo = (sysinfo_t *)(kp->ks_data);

        assert(kp->ks_data_size == sizeof (sysinfo_t));

        SAVE_UINT32(ksi, sysinfo, updates);
        SAVE_UINT32(ksi, sysinfo, runque);
        SAVE_UINT32(ksi, sysinfo, runocc);
        SAVE_UINT32(ksi, sysinfo, swpque);
        SAVE_UINT32(ksi, sysinfo, swpocc);
        SAVE_UINT32(ksi, sysinfo, waiting);
}

static void
save_vminfo(kstat_t *kp, ks_instance_t *ksi)
{
        vminfo_t        *vminfo = (vminfo_t *)(kp->ks_data);

        assert(kp->ks_data_size == sizeof (vminfo_t));

        SAVE_UINT64(ksi, vminfo, freemem);
        SAVE_UINT64(ksi, vminfo, swap_resv);
        SAVE_UINT64(ksi, vminfo, swap_alloc);
        SAVE_UINT64(ksi, vminfo, swap_avail);
        SAVE_UINT64(ksi, vminfo, swap_free);
        SAVE_UINT64(ksi, vminfo, updates);
}

static void
save_nfs(kstat_t *kp, ks_instance_t *ksi)
{
        struct mntinfo_kstat *mntinfo = (struct mntinfo_kstat *)(kp->ks_data);

        assert(kp->ks_data_size == sizeof (struct mntinfo_kstat));
        SAVE_STRING(ksi, mntinfo, mik_proto);
        SAVE_UINT32(ksi, mntinfo, mik_vers);
        SAVE_UINT32(ksi, mntinfo, mik_flags);
        SAVE_UINT32(ksi, mntinfo, mik_secmod);
        SAVE_UINT32(ksi, mntinfo, mik_curread);
        SAVE_UINT32(ksi, mntinfo, mik_curwrite);
        SAVE_INT32(ksi, mntinfo, mik_timeo);
        SAVE_INT32(ksi, mntinfo, mik_retrans);
        SAVE_UINT32(ksi, mntinfo, mik_acregmin);
        SAVE_UINT32(ksi, mntinfo, mik_acregmax);
        SAVE_UINT32(ksi, mntinfo, mik_acdirmin);
        SAVE_UINT32(ksi, mntinfo, mik_acdirmax);
        SAVE_UINT32_X(ksi, "lookup_srtt", mntinfo->mik_timers[0].srtt);
        SAVE_UINT32_X(ksi, "lookup_deviate", mntinfo->mik_timers[0].deviate);
        SAVE_UINT32_X(ksi, "lookup_rtxcur", mntinfo->mik_timers[0].rtxcur);
        SAVE_UINT32_X(ksi, "read_srtt", mntinfo->mik_timers[1].srtt);
        SAVE_UINT32_X(ksi, "read_deviate", mntinfo->mik_timers[1].deviate);
        SAVE_UINT32_X(ksi, "read_rtxcur", mntinfo->mik_timers[1].rtxcur);
        SAVE_UINT32_X(ksi, "write_srtt", mntinfo->mik_timers[2].srtt);
        SAVE_UINT32_X(ksi, "write_deviate", mntinfo->mik_timers[2].deviate);
        SAVE_UINT32_X(ksi, "write_rtxcur", mntinfo->mik_timers[2].rtxcur);
        SAVE_UINT32(ksi, mntinfo, mik_noresponse);
        SAVE_UINT32(ksi, mntinfo, mik_failover);
        SAVE_UINT32(ksi, mntinfo, mik_remap);
        SAVE_STRING(ksi, mntinfo, mik_curserver);
}

#ifdef __sparc
static void
save_sfmmu_global_stat(kstat_t *kp, ks_instance_t *ksi)
{
        struct sfmmu_global_stat *sfmmug =
            (struct sfmmu_global_stat *)(kp->ks_data);

        assert(kp->ks_data_size == sizeof (struct sfmmu_global_stat));

        SAVE_INT32(ksi, sfmmug, sf_tsb_exceptions);
        SAVE_INT32(ksi, sfmmug, sf_tsb_raise_exception);
        SAVE_INT32(ksi, sfmmug, sf_pagefaults);
        SAVE_INT32(ksi, sfmmug, sf_uhash_searches);
        SAVE_INT32(ksi, sfmmug, sf_uhash_links);
        SAVE_INT32(ksi, sfmmug, sf_khash_searches);
        SAVE_INT32(ksi, sfmmug, sf_khash_links);
        SAVE_INT32(ksi, sfmmug, sf_swapout);
        SAVE_INT32(ksi, sfmmug, sf_tsb_alloc);
        SAVE_INT32(ksi, sfmmug, sf_tsb_allocfail);
        SAVE_INT32(ksi, sfmmug, sf_tsb_sectsb_create);
        SAVE_INT32(ksi, sfmmug, sf_scd_1sttsb_alloc);
        SAVE_INT32(ksi, sfmmug, sf_scd_2ndtsb_alloc);
        SAVE_INT32(ksi, sfmmug, sf_scd_1sttsb_allocfail);
        SAVE_INT32(ksi, sfmmug, sf_scd_2ndtsb_allocfail);
        SAVE_INT32(ksi, sfmmug, sf_tteload8k);
        SAVE_INT32(ksi, sfmmug, sf_tteload64k);
        SAVE_INT32(ksi, sfmmug, sf_tteload512k);
        SAVE_INT32(ksi, sfmmug, sf_tteload4m);
        SAVE_INT32(ksi, sfmmug, sf_tteload32m);
        SAVE_INT32(ksi, sfmmug, sf_tteload256m);
        SAVE_INT32(ksi, sfmmug, sf_tsb_load8k);
        SAVE_INT32(ksi, sfmmug, sf_tsb_load4m);
        SAVE_INT32(ksi, sfmmug, sf_hblk_hit);
        SAVE_INT32(ksi, sfmmug, sf_hblk8_ncreate);
        SAVE_INT32(ksi, sfmmug, sf_hblk8_nalloc);
        SAVE_INT32(ksi, sfmmug, sf_hblk1_ncreate);
        SAVE_INT32(ksi, sfmmug, sf_hblk1_nalloc);
        SAVE_INT32(ksi, sfmmug, sf_hblk_slab_cnt);
        SAVE_INT32(ksi, sfmmug, sf_hblk_reserve_cnt);
```

```
        SAVE_INT32(ksi, sfmmug, sf_hblk_recurse_cnt);
        SAVE_INT32(ksi, sfmmug, sf_hblk_reserve_hit);
        SAVE_INT32(ksi, sfmmug, sf_get_free_success);
        SAVE_INT32(ksi, sfmmug, sf_get_free_throttle);
        SAVE_INT32(ksi, sfmmug, sf_get_free_fail);
        SAVE_INT32(ksi, sfmmug, sf_put_free_success);
        SAVE_INT32(ksi, sfmmug, sf_put_free_fail);
        SAVE_INT32(ksi, sfmmug, sf_pgcolor_conflict);
        SAVE_INT32(ksi, sfmmug, sf_uncache_conflict);
        SAVE_INT32(ksi, sfmmug, sf_unload_conflict);
        SAVE_INT32(ksi, sfmmug, sf_ism_uncache);
        SAVE_INT32(ksi, sfmmug, sf_ism_recache);
        SAVE_INT32(ksi, sfmmug, sf_recache);
        SAVE_INT32(ksi, sfmmug, sf_steal_count);
        SAVE_INT32(ksi, sfmmug, sf_pagesync);
        SAVE_INT32(ksi, sfmmug, sf_clrwrt);
        SAVE_INT32(ksi, sfmmug, sf_pagesync_invalid);
        SAVE_INT32(ksi, sfmmug, sf_kernel_xcalls);
        SAVE_INT32(ksi, sfmmug, sf_user_xcalls);
        SAVE_INT32(ksi, sfmmug, sf_tsb_grow);
        SAVE_INT32(ksi, sfmmug, sf_tsb_shrink);
        SAVE_INT32(ksi, sfmmug, sf_tsb_resize_failures);
        SAVE_INT32(ksi, sfmmug, sf_tsb_reloc);
        SAVE_INT32(ksi, sfmmug, sf_user_vtop);
        SAVE_INT32(ksi, sfmmug, sf_ctx_inv);
        SAVE_INT32(ksi, sfmmug, sf_tlb_reprog_pgsz);
        SAVE_INT32(ksi, sfmmug, sf_region_remap_demap);
        SAVE_INT32(ksi, sfmmug, sf_create_scd);
        SAVE_INT32(ksi, sfmmug, sf_join_scd);
        SAVE_INT32(ksi, sfmmug, sf_leave_scd);
        SAVE_INT32(ksi, sfmmug, sf_destroy_scd);
}
#endif

#ifdef __sparc
static void
save_sfmmu_tsbsize_stat(kstat_t *kp, ks_instance_t *ksi)
{
        struct sfmmu_tsbsize_stat *sfmmut;

        assert(kp->ks_data_size == sizeof (struct sfmmu_tsbsize_stat));
        sfmmut = (struct sfmmu_tsbsize_stat *)(kp->ks_data);

        SAVE_INT32(ksi, sfmmut, sf_tsbsz_8k);
        SAVE_INT32(ksi, sfmmut, sf_tsbsz_16k);
        SAVE_INT32(ksi, sfmmut, sf_tsbsz_32k);
        SAVE_INT32(ksi, sfmmut, sf_tsbsz_64k);
        SAVE_INT32(ksi, sfmmut, sf_tsbsz_128k);
        SAVE_INT32(ksi, sfmmut, sf_tsbsz_256k);
        SAVE_INT32(ksi, sfmmut, sf_tsbsz_512k);
        SAVE_INT32(ksi, sfmmut, sf_tsbsz_1m);
        SAVE_INT32(ksi, sfmmut, sf_tsbsz_2m);
        SAVE_INT32(ksi, sfmmut, sf_tsbsz_4m);
}
#endif

#ifdef __sparc
static void
save_simmstat(kstat_t *kp, ks_instance_t *ksi)
{
        uchar_t *simmstat;
        char    *simm_buf;
        char    *list = NULL;
        int     i;

        assert(kp->ks_data_size == sizeof (uchar_t) * SIMM_COUNT);
```

```
	for (i = 0, simmstat = (uchar_t *)(kp->ks_data); i < SIMM_COUNT - 1;
	    i++, simmstat++) {
		if (list == NULL) {
			(void) asprintf(&simm_buf, "%d,", *simmstat);
		} else {
			(void) asprintf(&simm_buf, "%s%d,", list, *simmstat);
			free(list);
		}
		list = simm_buf;
	}

	(void) asprintf(&simm_buf, "%s%d", list, *simmstat);
	SAVE_STRING_X(ksi, "status", simm_buf);
	free(list);
	free(simm_buf);
}
#endif

#ifdef __sparc
/*
 * Helper function for save_temperature().
 */
static char *
short_array_to_string(short *shortp, int len)
{
	char	*list = NULL;
	char	*list_buf;

	for (; len > 1; len--, shortp++) {
		if (list == NULL) {
			(void) asprintf(&list_buf, "%d,", *shortp);
		} else {
			(void) asprintf(&list_buf, "%s%d,", list, *shortp);
			free(list);
		}
		list = list_buf;
	}

	(void) asprintf(&list_buf, "%s%s", list, *shortp);
	free(list);
	return (list_buf);
}

static void
save_temperature(kstat_t *kp, ks_instance_t *ksi)
{
	struct temp_stats *temps = (struct temp_stats *)(kp->ks_data);
	char	*buf;
	int	n = 1;

	assert(kp->ks_data_size == sizeof (struct temp_stats));

	SAVE_UINT32(ksi, temps, index);

	buf = short_array_to_string(temps->l1, L1_SZ);
	SAVE_STRING_X(ksi, "l1", buf);
	free(buf);

	buf = short_array_to_string(temps->l2, L2_SZ);
	SAVE_STRING_X(ksi, "l2", buf);
	free(buf);

	buf = short_array_to_string(temps->l3, L3_SZ);
	SAVE_STRING_X(ksi, "l3", buf);
	free(buf);
```

```
	buf = short_array_to_string(temps->l4, L4_SZ);
	SAVE_STRING_X(ksi, "l4", buf);
	free(buf);

	buf = short_array_to_string(temps->l5, L5_SZ);
	SAVE_STRING_X(ksi, "l5", buf);
	free(buf);

	SAVE_INT32(ksi, temps, max);
	SAVE_INT32(ksi, temps, min);
	SAVE_INT32(ksi, temps, state);
	SAVE_INT32(ksi, temps, temp_cnt);
	SAVE_INT32(ksi, temps, shutdown_cnt);
	SAVE_INT32(ksi, temps, version);
	SAVE_INT32(ksi, temps, trend);
	SAVE_INT32(ksi, temps, override);
}
#endif

#ifdef __sparc
static void
save_temp_over(kstat_t *kp, ks_instance_t *ksi)
{
	short	*sh = (short *)(kp->ks_data);
	char	*value;

	assert(kp->ks_data_size == sizeof (short));

	(void) asprintf(&value, "%hu", *sh);
	SAVE_STRING_X(ksi, "override", value);
	free(value);
}
#endif

#ifdef __sparc
static void
save_ps_shadow(kstat_t *kp, ks_instance_t *ksi)
{
	uchar_t *uchar = (uchar_t *)(kp->ks_data);

	assert(kp->ks_data_size == SYS_PS_COUNT);

	SAVE_CHAR_X(ksi, "core_0", *uchar++);
	SAVE_CHAR_X(ksi, "core_1", *uchar++);
	SAVE_CHAR_X(ksi, "core_2", *uchar++);
	SAVE_CHAR_X(ksi, "core_3", *uchar++);
	SAVE_CHAR_X(ksi, "core_4", *uchar++);
	SAVE_CHAR_X(ksi, "core_5", *uchar++);
	SAVE_CHAR_X(ksi, "core_6", *uchar++);
	SAVE_CHAR_X(ksi, "core_7", *uchar++);
	SAVE_CHAR_X(ksi, "pps_0", *uchar++);
	SAVE_CHAR_X(ksi, "clk_33", *uchar++);
	SAVE_CHAR_X(ksi, "clk_50", *uchar++);
	SAVE_CHAR_X(ksi, "v5_p", *uchar++);
	SAVE_CHAR_X(ksi, "v12_p", *uchar++);
	SAVE_CHAR_X(ksi, "v5_aux", *uchar++);
	SAVE_CHAR_X(ksi, "v5_p_pch", *uchar++);
	SAVE_CHAR_X(ksi, "v12_p_pch", *uchar++);
	SAVE_CHAR_X(ksi, "v3_pch", *uchar++);
	SAVE_CHAR_X(ksi, "v5_pch", *uchar++);
	SAVE_CHAR_X(ksi, "p_fan", *uchar++);
}
#endif

#ifdef __sparc
```

```
static void
save_fault_list(kstat_t *kp, ks_instance_t *ksi)
{
        struct ft_list *fault;
        char    name[KSTAT_STRLEN + 7];
        int     i;

        for (i = 1, fault = (struct ft_list *)(kp->ks_data);
            i <= 999999 && i <= kp->ks_data_size / sizeof (struct ft_list);
            i++, fault++) {
                (void) snprintf(name, sizeof (name), "unit_%d", i);
                SAVE_INT32_X(ksi, name, fault->unit);
                (void) snprintf(name, sizeof (name), "type_%d", i);
                SAVE_INT32_X(ksi, name, fault->type);
                (void) snprintf(name, sizeof (name), "fclass_%d", i);
                SAVE_INT32_X(ksi, name, fault->fclass);
                (void) snprintf(name, sizeof (name), "create_time_%d", i);
                SAVE_HRTIME_X(ksi, name, fault->create_time);
                (void) snprintf(name, sizeof (name), "msg_%d", i);
                SAVE_STRING_X(ksi, name, faultp->msg);
        }
}
#endif

static void
save_named(kstat_t *kp, ks_instance_t *ksi)
{
        kstat_named_t *knp;
        int     n;

        for (n = kp->ks_ndata, knp = KSTAT_NAMED_PTR(kp); n > 0; n--, knp++) {
                switch (knp->data_type) {
                case KSTAT_DATA_CHAR:
                        nvpair_insert(ksi, knp->name,
                            (ks_value_t *)&knp->value, KSTAT_DATA_CHAR);
                        break;
                case KSTAT_DATA_INT32:
                        nvpair_insert(ksi, knp->name,
                            (ks_value_t *)&knp->value, KSTAT_DATA_INT32);
                        break;
                case KSTAT_DATA_UINT32:
                        nvpair_insert(ksi, knp->name,
                            (ks_value_t *)&knp->value, KSTAT_DATA_UINT32);
                        break;
                case KSTAT_DATA_INT64:
                        nvpair_insert(ksi, knp->name,
                            (ks_value_t *)&knp->value, KSTAT_DATA_INT64);
                        break;
                case KSTAT_DATA_UINT64:
                        nvpair_insert(ksi, knp->name,
                            (ks_value_t *)&knp->value, KSTAT_DATA_UINT64);
                        break;
                case KSTAT_DATA_STRING:
                        SAVE_STRING_X(ksi, knp->name, KSTAT_NAMED_STR_PTR(knp));
                        break;
                default:
                        assert(B_FALSE); /* Invalid data type */
                        break;
                }
        }
}

static void
save_intr(kstat_t *kp, ks_instance_t *ksi)
{
        kstat_intr_t *intr = KSTAT_INTR_PTR(kp);
```

```
        char    *intr_names[] = {"hard", "soft", "watchdog", "spurious",
            "multiple_service"};
        int     n;

        for (n = 0; n < KSTAT_NUM_INTRS; n++)
                SAVE_UINT32_X(ksi, intr_names[n], intr->intrs[n]);
}

static void
save_io(kstat_t *kp, ks_instance_t *ksi)
{
        kstat_io_t      *ksio = KSTAT_IO_PTR(kp);

        SAVE_UINT64(ksi, ksio, nread);
        SAVE_UINT64(ksi, ksio, nwritten);
        SAVE_UINT32(ksi, ksio, reads);
        SAVE_UINT32(ksi, ksio, writes);
        SAVE_HRTIME(ksi, ksio, wtime);
        SAVE_HRTIME(ksi, ksio, wlentime);
        SAVE_HRTIME(ksi, ksio, wlastupdate);
        SAVE_HRTIME(ksi, ksio, rtime);
        SAVE_HRTIME(ksi, ksio, rlentime);
        SAVE_HRTIME(ksi, ksio, rlastupdate);
        SAVE_UINT32(ksi, ksio, wcnt);
        SAVE_UINT32(ksi, ksio, rcnt);
}

static void
save_timer(kstat_t *kp, ks_instance_t *ksi)
{
        kstat_timer_t   *ktimer = KSTAT_TIMER_PTR(kp);

        SAVE_STRING(ksi, ktimer, name);
        SAVE_UINT64(ksi, ktimer, num_events);
        SAVE_HRTIME(ksi, ktimer, elapsed_time);
        SAVE_HRTIME(ksi, ktimer, min_time);
        SAVE_HRTIME(ksi, ktimer, max_time);
        SAVE_HRTIME(ksi, ktimer, start_time);
        SAVE_HRTIME(ksi, ktimer, stop_time);
}
#endif /* ! codereview */
```

```
***********************************************************
    6801 Fri Nov 30 19:01:28 2012
new/usr/src/cmd/stat/kstat/kstat.h
749 "/usr/bin/kstat" should be rewritten in C
Reviewed by: Garrett D'Amore <garrett@damore.org>
Reviewed by: Brendan Gregg <brendan.gregg@joyent.com>
***********************************************************
/*
 * 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.
 * Copyright 2012 David Hoeppner.  All rights reserved.
 */

#ifndef _STAT_KSTAT_H
#define _STAT_KSTAT_H

/*
 * Structures needed by the kstat reader functions.
 */
#include <sys/var.h>
#include <sys/utsname.h>
#include <sys/sysinfo.h>
#include <sys/flock.h>
#include <sys/dnlc.h>
#include <nfs/nfs.h>
#include <nfs/nfs_clnt.h>

#ifdef __sparc
#include <vm/hat_sfmmu.h>
#include <sys/simmstat.h>
#include <sys/sysctrl.h>
#include <sys/fhc.h>
#endif

#define KSTAT_DATA_HRTIME       (KSTAT_DATA_STRING + 1)

typedef union ks_value {
        char            c[16];
        int32_t         i32;
        uint32_t        ui32;
        struct {
                union {
                        char    *ptr;
                        char    __pad[8];
                } addr;
                uint32_t        len;
        } str;
```

```
        int64_t         i64;
        uint64_t        ui64;
} ks_value_t;

#define SAVE_HRTIME(I, S, N)                            \
{                                                       \
        ks_value_t v;                                   \
        v.ui64 = S->N;                                  \
        nvpair_insert(I, #N, &v, KSTAT_DATA_UINT64);    \
}

#define SAVE_INT32(I, S, N)                             \
{                                                       \
        ks_value_t v;                                   \
        v.i32 = S->N;                                   \
        nvpair_insert(I, #N, &v, KSTAT_DATA_INT32);     \
}

#define SAVE_UINT32(I, S, N)                            \
{                                                       \
        ks_value_t v;                                   \
        v.ui32 = S->N;                                  \
        nvpair_insert(I, #N, &v, KSTAT_DATA_UINT32);    \
}

#define SAVE_INT64(I, S, N)                             \
{                                                       \
        ks_value_t v;                                   \
        v.i64 = S->N;                                   \
        nvpair_insert(I, #N, &v, KSTAT_DATA_INT64);     \
}

#define SAVE_UINT64(I, S, N)                            \
{                                                       \
        ks_value_t v;                                   \
        v.ui64 = S->N;                                  \
        nvpair_insert(I, #N, &v, KSTAT_DATA_UINT64);    \
}

/*
 * We dont want const "strings" because we free
 * the instances later.
 */
#define SAVE_STRING(I, S, N)                            \
{                                                       \
        ks_value_t v;                                   \
        v.str.addr.ptr = safe_strdup(S->N);             \
        v.str.len = strlen(S->N);                       \
        nvpair_insert(I, #N, &v, KSTAT_DATA_STRING);    \
}

#define SAVE_HRTIME_X(I, N, V)                          \
{                                                       \
        ks_value_t v;                                   \
        v.ui64 = V;                                     \
        nvpair_insert(I, N, &v, KSTAT_DATA_HRTIME);     \
}

#define SAVE_INT32_X(I, N, V)                           \
{                                                       \
        ks_value_t v;                                   \
        v.i32 = V;                                      \
        nvpair_insert(I, N, &v, KSTAT_DATA_INT32);      \
}
```

```
#define SAVE_UINT32_X(I, N, V)                          \
{                                                       \
        ks_value_t v;                                   \
        v.ui32 = V;                                     \
        nvpair_insert(I, N, &v, KSTAT_DATA_UINT32);     \
}

#define SAVE_UINT64_X(I, N, V)                          \
{                                                       \
        ks_value_t v;                                   \
        v.ui64 = V;                                     \
        nvpair_insert(I, N, &v, KSTAT_DATA_UINT64);     \
}

#define SAVE_STRING_X(I, N, V)                          \
{                                                       \
        ks_value_t v;                                   \
        v.str.addr.ptr = safe_strdup(V);                \
        v.str.len = strlen(V);                          \
        nvpair_insert(I, N, &v, KSTAT_DATA_STRING);     \
}

#define SAVE_CHAR_X(I, N, V)                            \
{                                                       \
        ks_value_t v;                                   \
        asprintf(&v.str.addr.ptr, "%c", V);             \
        v.str.len = 1;                                  \
        nvpair_insert(I, N, &v, KSTAT_DATA_STRING);     \
}

#define DFLT_FMT                                        \
        "module: %-30.30s  instance: %-6d\n"            \
        "name:   %-30.30s  class:    %-.30s\n"

#define JSON_FMT                                        \
        "{\n\t\"module\": \"%s\",\n"                    \
        "\t\"instance\": %d,\n"                         \
        "\t\"name\": \"%s\",\n"                         \
        "\t\"class\": \"%s\",\n"                        \
        "\t\"type\": %d,\n"

#define KS_DFMT "\t%-30s  "
#define KS_JFMT "\t\t\"%s\": "
#define KS_PFMT "%s:%d:%s:%s"

typedef struct ks_instance {
        list_node_t     ks_next;
        char            ks_name[KSTAT_STRLEN];
        char            ks_module[KSTAT_STRLEN];
        char            ks_class[KSTAT_STRLEN];
        int             ks_instance;
        uchar_t         ks_type;
        hrtime_t        ks_snaptime;
        list_t          ks_nvlist;
} ks_instance_t;

typedef struct ks_nvpair {
        list_node_t     nv_next;
        char            name[KSTAT_STRLEN];
        uchar_t         data_type;
        ks_value_t      value;
} ks_nvpair_t;

typedef struct ks_selector {
        list_node_t ks_next;
        char    *ks_module;
```

```
        char    *ks_instance;
        char    *ks_name;
        char    *ks_statistic;
} ks_selector_t;

static void     usage(void);
static int      compare_instances(ks_instance_t *, ks_instance_t *);
static void     nvpair_insert(ks_instance_t *, char *, ks_value_t *, uchar_t);
static ks_selector_t    *new_selector(void);
static void     ks_instances_read(kstat_ctl_t *);
static void     ks_value_print(ks_nvpair_t *);
static void     ks_instance_print(ks_instance_t *, ks_nvpair_t *);
static void     ks_instances_print(void);
static char     *ks_safe_strdup(char *);
static void     ks_sleep_until(hrtime_t *, hrtime_t, int, int *);

/* Raw kstat readers */
static void     save_cpu_stat(kstat_t *, ks_instance_t *);
static void     save_var(kstat_t *, ks_instance_t *);
static void     save_ncstats(kstat_t *, ks_instance_t *);
static void     save_sysinfo(kstat_t *, ks_instance_t *);
static void     save_vminfo(kstat_t *, ks_instance_t *);
static void     save_nfs(kstat_t *, ks_instance_t *);
#ifdef __sparc
static void     save_sfmmu_global_stat(kstat_t *, ks_instance_t *);
static void     save_sfmmu_tsbsize_stat(kstat_t *, ks_instance_t *);
static void     save_simmstat(kstat_t *, ks_instance_t *);
/* Helper function for save_temperature() */
static char     *short_array_to_string(short *, int);
static void     save_temperature(kstat_t *, ks_instance_t *);
static void     save_temp_over(kstat_t *, ks_instance_t *);
static void     save_ps_shadow(kstat_t *, ks_instance_t *);
static void     save_fault_list(kstat_t *, ks_instance_t *);
#endif

/* Named kstat readers */
static void     save_named(kstat_t *, ks_instance_t *);
static void     save_intr(kstat_t *, ks_instance_t *);
static void     save_io(kstat_t *, ks_instance_t *);
static void     save_timer(kstat_t *, ks_instance_t *);

/* Typedef for raw kstat reader functions */
typedef void    (*kstat_raw_reader_t)(kstat_t *, ks_instance_t *);

static struct {
        kstat_raw_reader_t fn;
        char *name;
} ks_raw_lookup[] = {
        /* Function name                kstat name              */
        {save_cpu_stat,                 "cpu_stat:cpu_stat"},
        {save_var,                      "unix:var"},
        {save_ncstats,                  "unix:ncstats"},
        {save_sysinfo,                  "unix:sysinfo"},
        {save_vminfo,                   "unix:vminfo"},
        {save_nfs,                      "nfs:mntinfo"},
#ifdef __sparc
        {save_sfmmu_global_stat,        "unix:sfmmu_global_stat"},
        {save_sfmmu_tsbsize_stat,       "unix:sfmmu_tsbsize_stat"},
        {save_simmstat,                 "unix:simm-status"},
        {save_temperature,              "unix:temperature"},
        {save_temp_over,                "unix:temperature override"},
        {save_ps_shadow,                "unix:ps_shadow"},
        {save_fault_list,               "unix:fault_list"},
#endif
        {NULL, NULL},
};
```

```
static kstat_raw_reader_t       lookup_raw_kstat_fn(char *, char *);

#endif /* _STAT_KSTAT_H */
#endif /* ! codereview */
```

```
**********************************************************
    8929 Fri Nov 30 19:01:28 2012
new/usr/src/man/man1m/kstat.1m
749 "/usr/bin/kstat" should be rewritten in C
Reviewed by: Garrett D'Amore <garrett@damore.org>
Reviewed by: Brendan Gregg <brendan.gregg@joyent.com>
**********************************************************
'\" te
.\" Copyright (c) 2000, Sun Microsystems, Inc. All Rights Reserved
.\" The contents of this file are subject to the terms of the Common Development
.\"  See the License for the specific language governing permissions and limitat
.\" the fields enclosed by brackets "[]" replaced with your own identifying info
.TH KSTAT 1M "Nov 22, 2012"
.TH KSTAT 1M "Mar 23, 2009"
.SH NAME
kstat \- display kernel statistics
.SH SYNOPSIS
.LP
.nf
\fBkstat\fR [\fB-Cjlpq\fR] [\fB-T\fR u | d ] [\fB-c\fR \fIclass\fR] [\fB-m\fR \f
\fBkstat\fR [\fB-lpq\fR] [\fB-T\fR u | d ] [\fB-c\fR \fIclass\fR] [\fB-m\fR \fIm
     [\fB-i\fR \fIinstance\fR] [\fB-n\fR \fIname\fR] [\fB-s\fR \fIstatistic\fR]
     [interval [count]]
.fi

.LP
.nf
\fBkstat\fR [\fB-Cjlpq\fR] [\fB-T\fR u | d ] [\fB-c\fR \fIclass\fR]
\fBkstat\fR [\fB-lpq\fR] [\fB-T\fR u | d ] [\fB-c\fR \fIclass\fR]
     [\fImodule\fR:\fIinstance\fR:\fIname\fR:\fIstatistic\fR]...
     [interval [count]]
.fi

.SH DESCRIPTION
.sp
.LP
The \fBkstat\fR utility examines the available kernel statistics, or kstats, on
the system and reports those statistics which match the criteria specified on
the command line. Each matching statistic is printed with its module, instance,
and name fields, as well as its actual value.
.sp
.LP
Kernel statistics may be published by various kernel subsystems, such as
drivers or loadable modules; each kstat has a module field that denotes its
publisher. Since each module might have countable entities (such as multiple
disks associated with the \fBsd\fR(7D) driver) for which it wishes to report
statistics, the kstat also has an instance field to index the statistics for
each entity; kstat instances are numbered starting from zero. Finally, the
kstat is given a name unique within its module.
.sp
.LP
Each kstat may be a special kstat type, an array of name-value pairs, or raw
data. In the name-value case, each reported value is given a label, which we
refer to as the statistic. Known raw and special kstats are given statistic
labels for each of their values by \fBkstat\fR; thus, all published values can
be referenced as \fImodule\fR:\fIinstance\fR:\fIname\fR:\fIstatistic\fR.
.sp
.LP
When invoked without any module operands or options, kstat will match all
defined statistics on the system. Example invocations are provided below. All
times are displayed as fractional seconds since system boot.
.SH OPTIONS
.sp
.LP
The tests specified by the following options are logically ANDed, and all
matching kstats will be selected. A regular expression containing shell
```

```
metacharacters must be protected from the shell by enclosing it with the
appropriate quotes.
.sp
.LP
The argument for the \fB-c\fR, \fB-i\fR, \fB-m\fR, \fB-n\fR, and \fB-s\fR
options may be specified as a shell glob pattern.
.sp
.ne 2
.na
\fB\fB-C\fR\fR
.ad
.RS 16n
Displays output in parseable format with a colon as separator.
.RE

options may be specified as a shell glob pattern, or a Perl regular expression
enclosed in '/' characters.
.sp
.ne 2
.na
\fB\fB-c\fR \fIclass\fR\fR
.ad
.RS 16n
Displays only kstats that match the specified class. \fIclass\fR is a
kernel-defined string which classifies the "type" of the kstat.
.RE

.sp
.ne 2
.na
\fB\fB-i\fR \fIinstance\fR\fR
.ad
.RS 16n
Displays only kstats that match the specified instance.
.RE

.sp
.ne 2
.na
\fB\fB-j\fR\fR
.ad
.RS 16n
Displays output in JSON format.
.RE

.sp
.ne 2
.na
#endif /* ! codereview */
\fB\fB-l\fR\fR
.ad
.RS 16n
Lists matching kstat names without displaying values.
.RE

.sp
.ne 2
.na
\fB\fB-m\fR \fImodule\fR\fR
.ad
.RS 16n
Displays only kstats that match the specified module.
.RE

.sp
.ne 2
```

```
.na
\fB\fB-n\fR \fIname\fR\fR
.ad
.RS 16n
Displays only kstats that match the specified name.
.RE

.sp
.ne 2
.na
\fB\fB-p\fR\fR
.ad
.RS 16n
Displays output in parseable format. All example output in this document is
given in this format. If this option is not specified, \fBkstat\fR produces
output in a human-readable, table format.
.RE

.sp
.ne 2
.na
\fB\fB-q\fR\fR
.ad
.RS 16n
Displays no output, but return appropriate exit status for matches against
given criteria.
.RE

.sp
.ne 2
.na
\fB\fB-s\fR \fIstatistic\fR\fR
.ad
.RS 16n
Displays only kstats that match the specified statistic.
.RE

.sp
.ne 2
.na
\fB\fB-T\fR d | u\fR
.ad
.RS 16n
Displays a time stamp before each statistics block, either in \fBdate\fR(1)
format (\fBd\fR) or as an alphanumeric representation of the value returned by
\fBtime\fR(2) (\fBu\fR).
.RE

.SH OPERANDS
.sp
.LP
The following operands are supported:
.sp
.ne 2
.na
\fB\fImodule\fR:\fIinstance\fR:\fIname\fR:\fIstatistic\fR\fR
.ad
.sp .6
.RS 4n
Alternate method of specifying module, instance, name, and statistic as
described above. Each of the module, instance, name, or statistic specifiers
may be a shell glob pattern.
It is possible to use both specifier types within a single operand.
may be a shell glob pattern or a Perl regular expression enclosed by '/'
characters. It is possible to use both specifier types within a single operand.
Leaving a specifier empty is equivalent to using the '*' glob pattern for that
```

```
specifier.
.RE

.sp
.ne 2
.na
\fB\fIinterval\fR\fR
.ad
.sp .6
.RS 4n
The number of seconds between reports.
.RE

.sp
.ne 2
.na
\fB\fIcount\fR\fR
.ad
.sp .6
.RS 4n
The number of reports to be printed.
.RE

.SH EXAMPLES
.sp
.LP
In the following examples, all the command lines in a block produce the same
output, as shown immediately below. The exact statistics and values will of
course vary from machine to machine.
.LP
\fBExample 1 \fRUsing the \fBkstat\fR Command
.sp
.in +2
.nf
example$ \fBkstat -p -m unix -i 0 -n system_misc -s 'avenrun*'\fR
example$ \fBkstat -p -s 'avenrun*'\fR
example$ \fBkstat -p 'unix:0:system_misc:avenrun*'\fR
example$ \fBkstat -p ':::avenrun*'\fR
example$ \fBkstat -p ':::/^avenrun_\ed+min$/'\fR

unix:0:system_misc:avenrun_15min        3
unix:0:system_misc:avenrun_1min 4
unix:0:system_misc:avenrun_5min 2
.fi
.in -2
.sp

.LP
\fBExample 2 \fRUsing the \fBkstat\fR Command
.sp
.in +2
.nf
example$ \fBkstat -p -m cpu_stat -s 'intr*'\fR
example$ \fBkstat -p 'cpu_stat:::intr*'\fR
example$ \fBkstat -p cpu_stat:::/^intr/\fR

cpu_stat:0:cpu_stat0:intr       29682330
cpu_stat:0:cpu_stat0:intrblk    87
cpu_stat:0:cpu_stat0:intrthread 15054222
cpu_stat:1:cpu_stat1:intr       426073
cpu_stat:1:cpu_stat1:intrblk    51
cpu_stat:1:cpu_stat1:intrthread 289668
cpu_stat:2:cpu_stat2:intr       134160
cpu_stat:2:cpu_stat2:intrblk    0
cpu_stat:2:cpu_stat2:intrthread 131
cpu_stat:3:cpu_stat3:intr       196566
```

```
cpu_stat:3:cpu_stat3:intrblk    30
cpu_stat:3:cpu_stat3:intrthread 59626
.fi
.in -2
.sp

.LP
\fBExample 3 \fRUsing the \fBkstat\fR Command
.sp
.in +2
.nf
example$ \fBkstat -p :::state ':::avenrun*'\fR
example$ \fBkstat -p :::state :::/^avenrun/\fR

cpu_info:0:cpu_info0:state      on-line
cpu_info:1:cpu_info1:state      on-line
cpu_info:2:cpu_info2:state      on-line
cpu_info:3:cpu_info3:state      on-line
unix:0:system_misc:avenrun_15min        4
unix:0:system_misc:avenrun_1min 10
unix:0:system_misc:avenrun_5min 3
.fi
.in -2
.sp

.LP
\fBExample 4 \fRUsing the \fBkstat\fR Command
.sp
.in +2
.nf
example$ \fBkstat -p 'unix:0:system_misc:avenrun*' 1 3\fR
unix:0:system_misc:avenrun_15min        15
unix:0:system_misc:avenrun_1min 11
unix:0:system_misc:avenrun_5min 21

unix:0:system_misc:avenrun_15min        15
unix:0:system_misc:avenrun_1min 11
unix:0:system_misc:avenrun_5min 21

unix:0:system_misc:avenrun_15min        15
unix:0:system_misc:avenrun_1min 11
unix:0:system_misc:avenrun_5min 21
.fi
.in -2
.sp

.LP
\fBExample 5 \fRUsing the \fBkstat\fR Command
.sp
.in +2
.nf
example$ \fBkstat -p -T d 'unix:0:system_misc:avenrun*' 5 2\fR
Thu Jul 22 19:39:50 1999
unix:0:system_misc:avenrun_15min        12
unix:0:system_misc:avenrun_1min 0
unix:0:system_misc:avenrun_5min 11

Thu Jul 22 19:39:55 1999
unix:0:system_misc:avenrun_15min        12
unix:0:system_misc:avenrun_1min 0
unix:0:system_misc:avenrun_5min 11
.fi
.in -2
.sp

.LP
```

```
\fBExample 6 \fRUsing the \fBkstat\fR Command
.sp
.in +2
.nf
example$ \fBkstat -p -T u 'unix:0:system_misc:avenrun*'\fR
932668656
unix:0:system_misc:avenrun_15min        14
unix:0:system_misc:avenrun_1min 5
unix:0:system_misc:avenrun_5min 18
.fi
.in -2
.sp

.SH EXIT STATUS
.sp
.LP
The following exit values are returned:
.sp
.ne 2
.na
\fB\fB0\fR\fR
.ad
.RS 5n
One or more statistics were matched.
.RE

.sp
.ne 2
.na
\fB\fB1\fR\fR
.ad
.RS 5n
No statistics were matched.
.RE

.sp
.ne 2
.na
\fB\fB2\fR\fR
.ad
.RS 5n
Invalid command line options were specified.
.RE

.sp
.ne 2
.na
\fB\fB3\fR\fR
.ad
.RS 5n
A fatal error occurred.
.RE

.SH FILES
.sp
.ne 2
.na
\fB\fB/dev/kstat\fR\fR
.ad
.RS 14n
kernel statistics driver
.RE

.SH SEE ALSO
.sp
.LP
```

```
\fBdate\fR(1), \fBsh\fR(1), \fBtime\fR(2), \fBgmatch\fR(3GEN),
\fBkstat\fR(3KSTAT), \fBattributes\fR(5), \fBkstat\fR(7D), \fBsd\fR(7D),
\fBkstat\fR(9S)
.SH NOTES
.sp
.LP
If the pattern argument contains glob metacharacters which are also
If the pattern argument contains glob or Perl RE metacharacters which are also
shell metacharacters, it will be necessary to enclose the pattern with
appropriate shell quotes.
```