```
**********************************************************
    1267 Fri Jan 11 07:50:22 2013
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 Jan 11 07:50:22 2013
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 */
```

```
**********************************************************
   36691 Fri Jan 11 07:50:22 2013
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) 2013 David Hoeppner. All rights reserved.
 * Copyright 2013 Nexenta Systems, Inc.  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 replaced with extended REs bracketed by '/'
 *      - 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 ks_pattern_t     g_ks_class = {"*", 0};

/* 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.
```

```
126          */
127         while ((c = getopt(argc, argv, "h?CqjlpT:m:i:n:s:c:")) != EOF)
128                 switch (c) {
129                 case 'h':
130                 case '?':
131                         usage();
132                         exit(0);
133                         break;
134                 case 'C':
135                         g_pflg = g_cflg = B_TRUE;
136                         break;
137                 case 'q':
138                         g_qflg = B_TRUE;
139                         break;
140                 case 'j':
141                         g_jflg = B_TRUE;
142                         break;
143                 case 'l':
144                         g_pflg = g_lflg = B_TRUE;
145                         break;
146                 case 'p':
147                         g_pflg = B_TRUE;
148                         break;
149                 case 'T':
150                         switch (*optarg) {
151                         case 'd':
152                                 g_timestamp_fmt = DDATE;
153                                 break;
154                         case 'u':
155                                 g_timestamp_fmt = UDATE;
156                                 break;
157                         default:
158                                 errflg = B_TRUE;
159                         }
160                         break;
161                 case 'm':
162                         nselflg = B_TRUE;
163                         nselector->ks_module.pstr =
164                             (char *)ks_safe_strdup(optarg);
165                         break;
166                 case 'i':
167                         nselflg = B_TRUE;
168                         nselector->ks_instance.pstr =
169                             (char *)ks_safe_strdup(optarg);
170                         break;
171                 case 'n':
172                         nselflg = B_TRUE;
173                         nselector->ks_name.pstr =
174                             (char *)ks_safe_strdup(optarg);
175                         break;
176                 case 's':
177                         nselflg = B_TRUE;
178                         nselector->ks_statistic.pstr =
179                             (char *)ks_safe_strdup(optarg);
180                         break;
181                 case 'c':
182                         g_ks_class.pstr =
183                             (char *)ks_safe_strdup(optarg);
184                         break;
185                 default:
186                         errflg = B_TRUE;
187                         break;
188                 }

190         if (g_qflg && (g_jflg || g_pflg)) {
191                 (void) fprintf(stderr, gettext(
```



```
192                     "-q and -lpj are mutually exclusive\n"));
193                 errflg = B_TRUE;
194         }

196         if (errflg) {
197                 usage();
198                 exit(2);
199         }

201         argc -= optind;
202         argv += optind;

204         /*
205          * Consume the rest of the command line. Parsing the
206          * unnamed command line arguments.
207          */
208         while (argc--) {
209                 errno = 0;
210                 tmp = strtoul(*argv, &q, 10);
211                 if (tmp == ULONG_MAX && errno == ERANGE) {
212                         if (n == 0) {
213                                 (void) fprintf(stderr, gettext(
214                                     "Interval is too large\n"));
215                         } else if (n == 1) {
216                                 (void) fprintf(stderr, gettext(
217                                     "Count is too large\n"));
218                         }
219                         usage();
220                         exit(2);
221                 }

223                 if (errno != 0 || *q != '\0') {
224                         m = 0;
225                         uselector = new_selector();
226                         while ((q = (char *)strsep(argv, ":")) != NULL) {
227                                 m++;
228                                 if (m > 4) {
229                                         free(uselector);
230                                         usage();
231                                         exit(2);
232                                 }

234                                 if (*q != '\0') {
235                                         switch (m) {
236                                         case 1:
237                                                 uselector->ks_module.pstr =
238                                                     (char *)ks_safe_strdup(q);
239                                                 break;
240                                         case 2:
241                                                 uselector->ks_instance.pstr =
242                                                     (char *)ks_safe_strdup(q);
243                                                 break;
244                                         case 3:
245                                                 uselector->ks_name.pstr =
246                                                     (char *)ks_safe_strdup(q);
247                                                 break;
248                                         case 4:
249                                                 uselector->ks_statistic.pstr =
250                                                     (char *)ks_safe_strdup(q);
251                                                 break;
252                                         default:
253                                                 assert(B_FALSE);
254                                         }
255                                 }
256                         }
```

```
258                         if (m < 4) {
259                                 free(uselector);
260                                 usage();
261                                 exit(2);
262                         }

264                         uselflg = B_TRUE;
265                         list_insert_tail(&selector_list, uselector);
266                 } else {
267                         if (tmp < 1) {
268                                 if (n == 0) {
269                                         (void) fprintf(stderr, gettext(
270                                             "Interval must be an "
271                                             "integer >= 1"));
272                                 } else if (n == 1) {
273                                         (void) fprintf(stderr, gettext(
274                                             "Count must be an integer >= 1"));
275                                 }
276                                 usage();
277                                 exit(2);
278                         } else {
279                                 if (n == 0) {
280                                         interval = tmp;
281                                         count = -1;
282                                 } else if (n == 1) {
283                                         count = tmp;
284                                 } else {
285                                         usage();
286                                         exit(2);
287                                 }
288                         }
289                         n++;
290                 }
291                 argv++;
292         }

294         /*
295          * Check if we founded a named selector on the cmdline.
296          */
297         if (uselflg) {
298                 if (nselflg) {
299                         (void) fprintf(stderr, gettext(
300                             "module:instance:name:statistic and "
301                             "-m -i -n -s are mutually exclusive"));
302                         usage();
303                         exit(2);
304                 } else {
305                         free(nselector);
306                 }
307         } else {
308                 list_insert_tail(&selector_list, nselector);
309         }

311         assert(!list_is_empty(&selector_list));

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

316         while ((kc = kstat_open()) == NULL) {
317                 if (errno == EAGAIN) {
318                         (void) poll(NULL, 0, 200);
319                 } else {
320                         perror("kstat_open");
321                         exit(3);
322                 }
323         }
```



```
325         if (count > 1) {
326                 if (signal(SIGCONT, cont_handler) == SIG_ERR) {
327                         (void) fprintf(stderr, gettext(
328                             "signal failed"));
329                         exit(3);
330                 }
331         }

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

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

340                 if (interval && count) {
341                         ks_sleep_until(&start_n, period_n, infinite_cycles,
342                             &caught_cont);
343                         (void) kstat_chain_update(kc);
344                         (void) putchar('\n');
345                 }
346         }

348         (void) kstat_close(kc);

350         return (g_matched);
351 }

353 /*
354  * Print usage.
355  */
356 static void
357 usage(void)
358 {
359         (void) fprintf(stderr, gettext(
360             "Usage:\n"
361             "kstat [ -Cjlpq ] [ -T d|u ] [ -c class ]\n"
362             "      [ -m module ] [ -i instance ] [ -n name ] [ -s statistic ]\n"
363             "      [ interval [ count ] ]\n"
364             "kstat [ -Cjlpq ] [ -T d|u ] [ -c class ]\n"
365             "      [ module:instance:name:statistic ... ]\n"
366             "      [ interval [ count ] ]\n"));
367 }

369 /*
370  * Sort compare function.
371  */
372 static int
373 compare_instances(ks_instance_t *l_arg, ks_instance_t *r_arg)
374 {
375         int     rval;

377         rval = strcasecmp(l_arg->ks_module, r_arg->ks_module);
378         if (rval == 0) {
379                 if (l_arg->ks_instance == r_arg->ks_instance) {
380                         return (strcasecmp(l_arg->ks_name, r_arg->ks_name));
381                 } else if (l_arg->ks_instance < r_arg->ks_instance) {
382                         return (-1);
383                 } else {
384                         return (1);
385                 }
386         } else {
387                 return (rval);
388         }
389 }
```

```
391 static char *
392 ks_safe_strdup(char *str)
393 {
394         char    *ret;
395
396         if (str == NULL) {
397                 return (NULL);
398         }
399
400         while ((ret = strdup(str)) == NULL) {
401                 if (errno == EAGAIN) {
402                         (void) poll(NULL, 0, 200);
403                 } else {
404                         perror("strdup");
405                         exit(3);
406                 }
407         }
408
409         return (ret);
410 }
411
412 static void
413 ks_sleep_until(hrtime_t *wakeup, hrtime_t interval, int forever,
414     int *caught_cont)
415 {
416         hrtime_t        now, pause, pause_left;
417         struct timespec pause_tv;
418         int             status;
419
420         now = gethrtime();
421         pause = *wakeup + interval - now;
422
423         if (pause <= 0 || pause < (interval / 4)) {
424                 if (forever || *caught_cont) {
425                         *wakeup = now + interval;
426                         pause = interval;
427                 } else {
428                         pause = interval / 2;
429                         *wakeup += interval;
430                 }
431         } else {
432                 *wakeup += interval;
433         }
434
435         if (pause < 1000) {
436                 return;
437         }
438
439         pause_left = pause;
440         do {
441                 pause_tv.tv_sec = pause_left / NANOSEC;
442                 pause_tv.tv_nsec = pause_left % NANOSEC;
443                 status = nanosleep(&pause_tv, (struct timespec *)NULL);
444                 if (status < 0) {
445                         if (errno == EINTR) {
446                                 now = gethrtime();
447                                 pause_left = *wakeup - now;
448                                 if (pause_left < 1000) {
449                                         return;
450                                 }
451                         } else {
452                                 perror("nanosleep");
453                                 exit(3);
454                         }
455                 }
```



```
456         } while (status != 0);
457 }
458
459 /*
460  * Inserts an instance in the per selector list.
461  */
462 static void
463 nvpair_insert(ks_instance_t *ksi, char *name, ks_value_t *value,
464     uchar_t data_type)
465 {
466         ks_nvpair_t     *instance;
467         ks_nvpair_t     *tmp;
468
469         instance = (ks_nvpair_t *)malloc(sizeof (ks_nvpair_t));
470         if (instance == NULL) {
471                 perror("malloc");
472                 exit(3);
473         }
474
475         (void) strlcpy(instance->name, name, KSTAT_STRLEN);
476         (void) memcpy(&instance->value, value, sizeof (ks_value_t));
477         instance->data_type = data_type;
478
479         tmp = list_head(&ksi->ks_nvlist);
480         while (tmp != NULL && strcasecmp(instance->name, tmp->name) > 0)
481                 tmp = list_next(&ksi->ks_nvlist, tmp);
482
483         list_insert_before(&ksi->ks_nvlist, tmp, instance);
484 }
485
486 /*
487  * Allocates a new all-matching selector.
488  */
489 static ks_selector_t *
490 new_selector(void)
491 {
492         ks_selector_t   *selector;
493
494         selector = (ks_selector_t *)malloc(sizeof (ks_selector_t));
495         if (selector == NULL) {
496                 perror("malloc");
497                 exit(3);
498         }
499
500         list_link_init(&selector->ks_next);
501
502         selector->ks_module.pstr = "*";
503         selector->ks_instance.pstr = "*";
504         selector->ks_name.pstr = "*";
505         selector->ks_statistic.pstr = "*";
506
507         return (selector);
508 }
509
510 /*
511  * This function was taken from the perl kstat module code - please
512  * see for further comments there.
513  */
514 static kstat_raw_reader_t
515 lookup_raw_kstat_fn(char *module, char *name)
516 {
517         char            key[KSTAT_STRLEN * 2];
518         register char   *f, *t;
519         int             n = 0;
520
521         for (f = module, t = key; *f != '\0'; f++, t++) {
```

```
522                 while (*f != '\0' && isdigit(*f))
523                         f++;
524                 *t = *f;
525         }
526         *t++ = ':';

528         for (f = name; *f != '\0'; f++, t++) {
529                 while (*f != '\0' && isdigit(*f))
530                         f++;
531                 *t = *f;
532         }
533         *t = '\0';

535         while (ks_raw_lookup[n].fn != NULL) {
536                 if (strncmp(ks_raw_lookup[n].name, key, strlen(key)) == 0)
537                         return (ks_raw_lookup[n].fn);
538                 n++;
539         }

541         return (0);
542 }

544 /*
545  * Match a string against a shell glob or extended regular expression.
546  */
547 static boolean_t
548 ks_match(const char *str, ks_pattern_t *pattern)
549 {
550         int     regcode;
551         char    *regstr;
552         char    *errbuf;
553         size_t  bufsz;

555         if (pattern->pstr != NULL && gmatch(pattern->pstr, "/*/") != 0) {
556                 /* All regex patterns are strdup'd copies */
557                 regstr = pattern->pstr + 1;
558                 *(strrchr(regstr, '/')) = '\0';

560                 regcode = regcomp(&pattern->preg, regstr,
561                     REG_EXTENDED | REG_NOSUB);
562                 if (regcode != 0) {
563                         bufsz = regerror(regcode, NULL, NULL, 0);
564                         if (bufsz != 0) {
565                                 errbuf = malloc(bufsz);
566                                 if (errbuf == NULL) {
567                                         perror("malloc");
568                                         exit(3);
569                                 }
570                                 (void) regerror(regcode, NULL, errbuf, bufsz);
571                                 (void) fprintf(stderr, "kstat: %s\n", errbuf);
572                         }
573                         usage();
574                         exit(2);
575                 }

577                 pattern->pstr = NULL;
578         }

580         if (pattern->pstr == NULL) {
581                 return (regexec(&pattern->preg, str, 0, NULL, 0) == 0);
582         }

584         return ((gmatch(str, pattern->pstr) != 0));
585 }

587 /*
```



```
588  * Iterate over all kernel statistics and save matches.
589  */
590 static void
591 ks_instances_read(kstat_ctl_t *kc)
592 {
593         kstat_raw_reader_t save_raw = NULL;
594         kid_t           id;
595         ks_selector_t   *selector;
596         ks_instance_t   *ksi;
597         ks_instance_t   *tmp;
598         kstat_t         *kp;
599         boolean_t       skip;

601         for (kp = kc->kc_chain; kp != NULL; kp = kp->ks_next) {
602                 /* Don't bother storing the kstat headers */
603                 if (strncmp(kp->ks_name, "kstat_", 6) == 0) {
604                         continue;
605                 }

607                 /* Don't bother storing raw stats we don't understand */
608                 if (kp->ks_type == KSTAT_TYPE_RAW) {
609                         save_raw = lookup_raw_kstat_fn(kp->ks_module,
610                             kp->ks_name);
611                         if (save_raw == NULL) {
612 #ifdef REPORT_UNKNOWN
613                                 (void) fprintf(stderr,
614                                     "Unknown kstat type %s:%d:%s - "
615                                     "%d of size %d\n", kp->ks_module,
616                                     kp->ks_instance, kp->ks_name,
617                                     kp->ks_ndata, kp->ks_data_size);
618 #endif
619                                 continue;
620                         }
621                 }

623                 /*
624                  * Iterate over the list of selectors and skip
625                  * instances we dont want. We filter for statistics
626                  * later, as we dont know them yet.
627                  */
628                 skip = B_TRUE;
629                 selector = list_head(&selector_list);
630                 while (selector != NULL) {
631                         if (ks_match(kp->ks_module, &selector->ks_module) ||
632                             ks_match(kp->ks_name, &selector->ks_name)) {
633                                 skip = B_FALSE;
634                                 break;
635                         }
636                         selector = list_next(&selector_list, selector);
637                 }

639                 if (skip) {
640                         continue;
641                 }

643                 /*
644                  * Allocate a new instance and fill in the values
645                  * we know so far.
646                  */
647                 ksi = (ks_instance_t *)malloc(sizeof (ks_instance_t));
648                 if (ksi == NULL) {
649                         perror("malloc");
650                         exit(3);
651                 }

653                 list_link_init(&ksi->ks_next);
```

```
655                 (void) strlcpy(ksi->ks_module, kp->ks_module, KSTAT_STRLEN);
656                 (void) strlcpy(ksi->ks_name, kp->ks_name, KSTAT_STRLEN);
657                 (void) strlcpy(ksi->ks_class, kp->ks_class, KSTAT_STRLEN);

659                 ksi->ks_instance = kp->ks_instance;
660                 ksi->ks_snaptime = kp->ks_snaptime;
661                 ksi->ks_type = kp->ks_type;

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

666                 SAVE_HRTIME_X(ksi, "crtime", kp->ks_crtime);
667                 SAVE_HRTIME_X(ksi, "snaptime", kp->ks_snaptime);
668                 if (g_pflg) {
669                         SAVE_STRING_X(ksi, "class", kp->ks_class);
670                 }

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

677                 list_insert_before(&instances_list, tmp, ksi);

679                 /* Read the actual statistics */
680                 id = kstat_read(kc, kp, NULL);
681                 if (id == -1) {
682 #ifdef REPORT_UNKNOWN
683                         perror("kstat_read");
684 #endif
685                         continue;
686                 }

688                 switch (kp->ks_type) {
689                 case KSTAT_TYPE_RAW:
690                         save_raw(kp, ksi);
691                         break;
692                 case KSTAT_TYPE_NAMED:
693                         save_named(kp, ksi);
694                         break;
695                 case KSTAT_TYPE_INTR:
696                         save_intr(kp, ksi);
697                         break;
698                 case KSTAT_TYPE_IO:
699                         save_io(kp, ksi);
700                         break;
701                 case KSTAT_TYPE_TIMER:
702                         save_timer(kp, ksi);
703                         break;
704                 default:
705                         assert(B_FALSE); /* Invalid type */
706                         break;
707                 }
708         }
709 }

711 /*
712  * Print the value of a name-value pair.
713  */
714 static void
715 ks_value_print(ks_nvpair_t *nvpair)
716 {
717         switch (nvpair->data_type) {
718         case KSTAT_DATA_CHAR:
719                 (void) fprintf(stdout, "%s", nvpair->value.c);
```



```
720                 break;
721         case KSTAT_DATA_INT32:
722                 (void) fprintf(stdout, "%d", nvpair->value.i32);
723                 break;
724         case KSTAT_DATA_UINT32:
725                 (void) fprintf(stdout, "%u", nvpair->value.ui32);
726                 break;
727         case KSTAT_DATA_INT64:
728                 (void) fprintf(stdout, "%lld", nvpair->value.i64);
729                 break;
730         case KSTAT_DATA_UINT64:
731                 (void) fprintf(stdout, "%llu", nvpair->value.ui64);
732                 break;
733         case KSTAT_DATA_STRING:
734                 (void) fprintf(stdout, "%s", KSTAT_NAMED_STR_PTR(nvpair));
735                 break;
736         case KSTAT_DATA_HRTIME:
737                 if (nvpair->value.ui64 == 0)
738                         (void) fprintf(stdout, "0");
739                 else
740                         (void) fprintf(stdout, "%.9f",
741                             nvpair->value.ui64 / 1000000000.0);
742                 break;
743         default:
744                 assert(B_FALSE);
745         }
746 }

748 /*
749  * Print a single instance.
750  */
751 static void
752 ks_instance_print(ks_instance_t *ksi, ks_nvpair_t *nvpair)
753 {
754         if (g_headerflg) {
755                 if (!g_pflg) {
756                         (void) fprintf(stdout, DFLT_FMT,
757                             ksi->ks_module, ksi->ks_instance,
758                             ksi->ks_name, ksi->ks_class);
759                 }
760                 g_headerflg = B_FALSE;
761         }

763         if (g_pflg) {
764                 (void) fprintf(stdout, KS_PFMT,
765                     ksi->ks_module, ksi->ks_instance,
766                     ksi->ks_name, nvpair->name);
767                 if (!g_lflg) {
768                         (void) putchar(g_cflg ? ':': '\t');
769                         ks_value_print(nvpair);
770                 }
771         } else {
772                 (void) fprintf(stdout, KS_DFMT, nvpair->name);
773                 ks_value_print(nvpair);
774         }

776         (void) putchar('\n');
777 }

779 /*
780  * Print a single instance in JSON format.
781  */
782 static void
783 ks_instance_print_json(ks_instance_t *ksi, ks_nvpair_t *nvpair)
784 {
785         if (g_headerflg) {
```

```
786                 (void) fprintf(stdout, JSON_FMT,
787                     ksi->ks_module, ksi->ks_instance,
788                     ksi->ks_name, ksi->ks_class,
789                     ksi->ks_type);

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

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

799                 g_headerflg = B_FALSE;
800         }

802         (void) fprintf(stdout, KS_JFMT, nvpair->name);
803         if (nvpair->data_type == KSTAT_DATA_STRING) {
804                 (void) putchar('\"');
805                 ks_value_print(nvpair);
806                 (void) putchar('\"');
807         } else {
808                 ks_value_print(nvpair);
809         }
810         if (nvpair != list_tail(&ksi->ks_nvlist))
811                 (void) putchar(',');

813         (void) putchar('\n');
814 }

816 /*
817  * Print all instances.
818  */
819 static void
820 ks_instances_print(void)
821 {
822         ks_selector_t   *selector;
823         ks_instance_t   *ksi, *ktmp;
824         ks_nvpair_t     *nvpair, *ntmp;
825         void            (*ks_print_fn)(ks_instance_t *, ks_nvpair_t *);
826         char            *ks_number;

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

831         if (g_jflg) {
832                 ks_print_fn = &ks_instance_print_json;
833                 (void) putchar('[');
834         } else {
835                 ks_print_fn = &ks_instance_print;
836         }

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

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

846                         (void) asprintf(&ks_number, "%d", ksi->ks_instance);
847                         if (!(ks_match(ksi->ks_module, &selector->ks_module) &&
848                             ks_match(ksi->ks_name, &selector->ks_name) &&
849                             ks_match(ks_number, &selector->ks_instance) &&
850                             ks_match(ksi->ks_class, &g_ks_class))) {
851                                 free(ks_number);
```



```
852                                 continue;
853                         }

855                         free(ks_number);

857                         /* Finally iterate over each statistic */
858                         g_headerflg = B_TRUE;
859                         for (nvpair = list_head(&ksi->ks_nvlist);
860                             nvpair != NULL;
861                             nvpair = list_next(&ksi->ks_nvlist, nvpair)) {
862                                 if (!ks_match(nvpair->name,
863                                     &selector->ks_statistic))
864                                         continue;

866                                 g_matched = 0;
867                                 if (!g_qflg)
868                                         (*ks_print_fn)(ksi, nvpair);
869                         }

871                         if (!g_headerflg) {
872                                 if (g_jflg) {
873                                         (void) fprintf(stdout, "\t}\n}");
874                                         if (ksi != list_tail(&instances_list))
875                                                 (void) putchar(',');
876                                 } else if (!g_pflg) {
877                                         (void) putchar('\n');
878                                 }
879                         }
880                 }

882                 selector = list_next(&selector_list, selector);
883         }

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

888         (void) fflush(stdout);

890         /* Free the instances list */
891         ksi = list_head(&instances_list);
892         while (ksi != NULL) {
893                 nvpair = list_head(&ksi->ks_nvlist);
894                 while (nvpair != NULL) {
895                         ntmp = nvpair;
896                         nvpair = list_next(&ksi->ks_nvlist, nvpair);
897                         list_remove(&ksi->ks_nvlist, ntmp);
898                         if (ntmp->data_type == KSTAT_DATA_STRING)
899                                 free(ntmp->value.str.addr.ptr);
900                         free(ntmp);
901                 }

903                 ktmp = ksi;
904                 ksi = list_next(&instances_list, ksi);
905                 list_remove(&instances_list, ktmp);
906                 list_destroy(&ktmp->ks_nvlist);
907                 free(ktmp);
908         }
909 }

911 static void
912 save_cpu_stat(kstat_t *kp, ks_instance_t *ksi)
913 {
914         cpu_stat_t      *stat;
915         cpu_sysinfo_t   *sysinfo;
916         cpu_syswait_t   *syswait;
917         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);
        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_UINT32(ksi, syswait, iowait);
        SAVE_UINT32(ksi, syswait, swap);
        SAVE_UINT32(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);
        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)
{
```

```
1050         struct ncstats  *ncstats = (struct ncstats *)(kp->ks_data);

1052         assert(kp->ks_data_size == sizeof (struct ncstats));

1054         SAVE_INT32(ksi, ncstats, hits);
1055         SAVE_INT32(ksi, ncstats, misses);
1056         SAVE_INT32(ksi, ncstats, enters);
1057         SAVE_INT32(ksi, ncstats, dbl_enters);
1058         SAVE_INT32(ksi, ncstats, long_enter);
1059         SAVE_INT32(ksi, ncstats, long_look);
1060         SAVE_INT32(ksi, ncstats, move_to_front);
1061         SAVE_INT32(ksi, ncstats, purges);
1062 }

1064 static void
1065 save_sysinfo(kstat_t *kp, ks_instance_t *ksi)
1066 {
1067         sysinfo_t       *sysinfo = (sysinfo_t *)(kp->ks_data);

1069         assert(kp->ks_data_size == sizeof (sysinfo_t));

1071         SAVE_UINT32(ksi, sysinfo, updates);
1072         SAVE_UINT32(ksi, sysinfo, runque);
1073         SAVE_UINT32(ksi, sysinfo, runocc);
1074         SAVE_UINT32(ksi, sysinfo, swpque);
1075         SAVE_UINT32(ksi, sysinfo, swpocc);
1076         SAVE_UINT32(ksi, sysinfo, waiting);
1077 }

1079 static void
1080 save_vminfo(kstat_t *kp, ks_instance_t *ksi)
1081 {
1082         vminfo_t        *vminfo = (vminfo_t *)(kp->ks_data);

1084         assert(kp->ks_data_size == sizeof (vminfo_t));

1086         SAVE_UINT64(ksi, vminfo, freemem);
1087         SAVE_UINT64(ksi, vminfo, swap_resv);
1088         SAVE_UINT64(ksi, vminfo, swap_alloc);
1089         SAVE_UINT64(ksi, vminfo, swap_avail);
1090         SAVE_UINT64(ksi, vminfo, swap_free);
1091         SAVE_UINT64(ksi, vminfo, updates);
1092 }

1094 static void
1095 save_nfs(kstat_t *kp, ks_instance_t *ksi)
1096 {
1097         struct mntinfo_kstat *mntinfo = (struct mntinfo_kstat *)(kp->ks_data);

1099         assert(kp->ks_data_size == sizeof (struct mntinfo_kstat));

1101         SAVE_STRING(ksi, mntinfo, mik_proto);
1102         SAVE_UINT32(ksi, mntinfo, mik_vers);
1103         SAVE_UINT32(ksi, mntinfo, mik_flags);
1104         SAVE_UINT32(ksi, mntinfo, mik_secmod);
1105         SAVE_UINT32(ksi, mntinfo, mik_curread);
1106         SAVE_UINT32(ksi, mntinfo, mik_curwrite);
1107         SAVE_INT32(ksi, mntinfo, mik_timeo);
1108         SAVE_INT32(ksi, mntinfo, mik_retrans);
1109         SAVE_UINT32(ksi, mntinfo, mik_acregmin);
1110         SAVE_UINT32(ksi, mntinfo, mik_acregmax);
1111         SAVE_UINT32(ksi, mntinfo, mik_acdirmin);
1112         SAVE_UINT32(ksi, mntinfo, mik_acdirmax);
1113         SAVE_UINT32_X(ksi, "lookup_srtt", mntinfo->mik_timers[0].srtt);
1114         SAVE_UINT32_X(ksi, "lookup_deviate", mntinfo->mik_timers[0].deviate);
1115         SAVE_UINT32_X(ksi, "lookup_rtxcur", mntinfo->mik_timers[0].rtxcur);
```



```
1116         SAVE_UINT32_X(ksi, "read_srtt", mntinfo->mik_timers[1].srtt);
1117         SAVE_UINT32_X(ksi, "read_deviate", mntinfo->mik_timers[1].deviate);
1118         SAVE_UINT32_X(ksi, "read_rtxcur", mntinfo->mik_timers[1].rtxcur);
1119         SAVE_UINT32_X(ksi, "write_srtt", mntinfo->mik_timers[2].srtt);
1120         SAVE_UINT32_X(ksi, "write_deviate", mntinfo->mik_timers[2].deviate);
1121         SAVE_UINT32_X(ksi, "write_rtxcur", mntinfo->mik_timers[2].rtxcur);
1122         SAVE_UINT32(ksi, mntinfo, mik_noresponse);
1123         SAVE_UINT32(ksi, mntinfo, mik_failover);
1124         SAVE_UINT32(ksi, mntinfo, mik_remap);
1125         SAVE_STRING(ksi, mntinfo, mik_curserver);
1126 }

1128 #ifdef __sparc
1129 static void
1130 save_sfmmu_global_stat(kstat_t *kp, ks_instance_t *ksi)
1131 {
1132         struct sfmmu_global_stat *sfmmug =
1133             (struct sfmmu_global_stat *)(kp->ks_data);

1135         assert(kp->ks_data_size == sizeof (struct sfmmu_global_stat));

1137         SAVE_INT32(ksi, sfmmug, sf_tsb_exceptions);
1138         SAVE_INT32(ksi, sfmmug, sf_tsb_raise_exception);
1139         SAVE_INT32(ksi, sfmmug, sf_pagefaults);
1140         SAVE_INT32(ksi, sfmmug, sf_uhash_searches);
1141         SAVE_INT32(ksi, sfmmug, sf_uhash_links);
1142         SAVE_INT32(ksi, sfmmug, sf_khash_searches);
1143         SAVE_INT32(ksi, sfmmug, sf_khash_links);
1144         SAVE_INT32(ksi, sfmmug, sf_swapout);
1145         SAVE_INT32(ksi, sfmmug, sf_tsb_alloc);
1146         SAVE_INT32(ksi, sfmmug, sf_tsb_allocfail);
1147         SAVE_INT32(ksi, sfmmug, sf_tsb_sectsb_create);
1148         SAVE_INT32(ksi, sfmmug, sf_scd_1sttsb_alloc);
1149         SAVE_INT32(ksi, sfmmug, sf_scd_2ndtsb_alloc);
1150         SAVE_INT32(ksi, sfmmug, sf_scd_1sttsb_allocfail);
1151         SAVE_INT32(ksi, sfmmug, sf_scd_2ndtsb_allocfail);
1152         SAVE_INT32(ksi, sfmmug, sf_tteload8k);
1153         SAVE_INT32(ksi, sfmmug, sf_tteload64k);
1154         SAVE_INT32(ksi, sfmmug, sf_tteload512k);
1155         SAVE_INT32(ksi, sfmmug, sf_tteload4m);
1156         SAVE_INT32(ksi, sfmmug, sf_tteload32m);
1157         SAVE_INT32(ksi, sfmmug, sf_tteload256m);
1158         SAVE_INT32(ksi, sfmmug, sf_tsb_load8k);
1159         SAVE_INT32(ksi, sfmmug, sf_tsb_load4m);
1160         SAVE_INT32(ksi, sfmmug, sf_hblk_hit);
1161         SAVE_INT32(ksi, sfmmug, sf_hblk8_ncreate);
1162         SAVE_INT32(ksi, sfmmug, sf_hblk8_nalloc);
1163         SAVE_INT32(ksi, sfmmug, sf_hblk1_ncreate);
1164         SAVE_INT32(ksi, sfmmug, sf_hblk1_nalloc);
1165         SAVE_INT32(ksi, sfmmug, sf_hblk_slab_cnt);
1166         SAVE_INT32(ksi, sfmmug, sf_hblk_reserve_cnt);
1167         SAVE_INT32(ksi, sfmmug, sf_hblk_recurse_cnt);
1168         SAVE_INT32(ksi, sfmmug, sf_hblk_reserve_hit);
1169         SAVE_INT32(ksi, sfmmug, sf_get_free_success);
1170         SAVE_INT32(ksi, sfmmug, sf_get_free_throttle);
1171         SAVE_INT32(ksi, sfmmug, sf_get_free_fail);
1172         SAVE_INT32(ksi, sfmmug, sf_put_free_success);
1173         SAVE_INT32(ksi, sfmmug, sf_put_free_fail);
1174         SAVE_INT32(ksi, sfmmug, sf_pgcolor_conflict);
1175         SAVE_INT32(ksi, sfmmug, sf_uncache_conflict);
1176         SAVE_INT32(ksi, sfmmug, sf_unload_conflict);
1177         SAVE_INT32(ksi, sfmmug, sf_ism_uncache);
1178         SAVE_INT32(ksi, sfmmug, sf_ism_recache);
1179         SAVE_INT32(ksi, sfmmug, sf_recache);
1180         SAVE_INT32(ksi, sfmmug, sf_steal_count);
1181         SAVE_INT32(ksi, sfmmug, sf_pagesync);
```

```
1182         SAVE_INT32(ksi, sfmmug, sf_clrwrt);
1183         SAVE_INT32(ksi, sfmmug, sf_pagesync_invalid);
1184         SAVE_INT32(ksi, sfmmug, sf_kernel_xcalls);
1185         SAVE_INT32(ksi, sfmmug, sf_user_xcalls);
1186         SAVE_INT32(ksi, sfmmug, sf_tsb_grow);
1187         SAVE_INT32(ksi, sfmmug, sf_tsb_shrink);
1188         SAVE_INT32(ksi, sfmmug, sf_tsb_resize_failures);
1189         SAVE_INT32(ksi, sfmmug, sf_tsb_reloc);
1190         SAVE_INT32(ksi, sfmmug, sf_user_vtop);
1191         SAVE_INT32(ksi, sfmmug, sf_ctx_inv);
1192         SAVE_INT32(ksi, sfmmug, sf_tlb_reprog_pgsz);
1193         SAVE_INT32(ksi, sfmmug, sf_region_remap_demap);
1194         SAVE_INT32(ksi, sfmmug, sf_create_scd);
1195         SAVE_INT32(ksi, sfmmug, sf_join_scd);
1196         SAVE_INT32(ksi, sfmmug, sf_leave_scd);
1197         SAVE_INT32(ksi, sfmmug, sf_destroy_scd);
1198 }
1199 #endif

1201 #ifdef __sparc
1202 static void
1203 save_sfmmu_tsbsize_stat(kstat_t *kp, ks_instance_t *ksi)
1204 {
1205         struct sfmmu_tsbsize_stat *sfmmut;

1207         assert(kp->ks_data_size == sizeof (struct sfmmu_tsbsize_stat));
1208         sfmmut = (struct sfmmu_tsbsize_stat *)(kp->ks_data);

1210         SAVE_INT32(ksi, sfmmut, sf_tsbsz_8k);
1211         SAVE_INT32(ksi, sfmmut, sf_tsbsz_16k);
1212         SAVE_INT32(ksi, sfmmut, sf_tsbsz_32k);
1213         SAVE_INT32(ksi, sfmmut, sf_tsbsz_64k);
1214         SAVE_INT32(ksi, sfmmut, sf_tsbsz_128k);
1215         SAVE_INT32(ksi, sfmmut, sf_tsbsz_256k);
1216         SAVE_INT32(ksi, sfmmut, sf_tsbsz_512k);
1217         SAVE_INT32(ksi, sfmmut, sf_tsbsz_1m);
1218         SAVE_INT32(ksi, sfmmut, sf_tsbsz_2m);
1219         SAVE_INT32(ksi, sfmmut, sf_tsbsz_4m);
1220 }
1221 #endif

1223 #ifdef __sparc
1224 static void
1225 save_simmstat(kstat_t *kp, ks_instance_t *ksi)
1226 {
1227         uchar_t *simmstat;
1228         char    *simm_buf;
1229         char    *list = NULL;
1230         int     i;

1232         assert(kp->ks_data_size == sizeof (uchar_t) * SIMM_COUNT);

1234         for (i = 0, simmstat = (uchar_t *)(kp->ks_data); i < SIMM_COUNT - 1;
1235             i++, simmstat++) {
1236                 if (list == NULL) {
1237                         (void) asprintf(&simm_buf, "%d,", *simmstat);
1238                 } else {
1239                         (void) asprintf(&simm_buf, "%s%d,", list, *simmstat);
1240                         free(list);
1241                 }
1242                 list = simm_buf;
1243         }

1245         (void) asprintf(&simm_buf, "%s%d", list, *simmstat);
1246         SAVE_STRING_X(ksi, "status", simm_buf);
1247         free(list);
```



```
1248         free(simm_buf);
1249 }
1250 #endif

1252 #ifdef __sparc
1253 /*
1254  * Helper function for save_temperature().
1255  */
1256 static char *
1257 short_array_to_string(short *shortp, int len)
1258 {
1259         char    *list = NULL;
1260         char    *list_buf;

1262         for (; len > 1; len--, shortp++) {
1263                 if (list == NULL) {
1264                         (void) asprintf(&list_buf, "%d,", *shortp);
1265                 } else {
1266                         (void) asprintf(&list_buf, "%s%d,", list, *shortp);
1267                         free(list);
1268                 }
1269                 list = list_buf;
1270         }

1272         (void) asprintf(&list_buf, "%s%s", list, *shortp);
1273         free(list);
1274         return (list_buf);
1275 }

1277 static void
1278 save_temperature(kstat_t *kp, ks_instance_t *ksi)
1279 {
1280         struct temp_stats *temps = (struct temp_stats *)(kp->ks_data);
1281         char    *buf;
1282         int     n = 1;

1284         assert(kp->ks_data_size == sizeof (struct temp_stats));

1286         SAVE_UINT32(ksi, temps, index);

1288         buf = short_array_to_string(temps->l1, L1_SZ);
1289         SAVE_STRING_X(ksi, "l1", buf);
1290         free(buf);

1292         buf = short_array_to_string(temps->l2, L2_SZ);
1293         SAVE_STRING_X(ksi, "l2", buf);
1294         free(buf);

1296         buf = short_array_to_string(temps->l3, L3_SZ);
1297         SAVE_STRING_X(ksi, "l3", buf);
1298         free(buf);

1300         buf = short_array_to_string(temps->l4, L4_SZ);
1301         SAVE_STRING_X(ksi, "l4", buf);
1302         free(buf);

1304         buf = short_array_to_string(temps->l5, L5_SZ);
1305         SAVE_STRING_X(ksi, "l5", buf);
1306         free(buf);

1308         SAVE_INT32(ksi, temps, max);
1309         SAVE_INT32(ksi, temps, min);
1310         SAVE_INT32(ksi, temps, state);
1311         SAVE_INT32(ksi, temps, temp_cnt);
1312         SAVE_INT32(ksi, temps, shutdown_cnt);
1313         SAVE_INT32(ksi, temps, version);
```

```
1314         SAVE_INT32(ksi, temps, trend);
1315         SAVE_INT32(ksi, temps, override);
1316 }
1317 #endif

1319 #ifdef __sparc
1320 static void
1321 save_temp_over(kstat_t *kp, ks_instance_t *ksi)
1322 {
1323         short   *sh = (short *)(kp->ks_data);
1324         char    *value;

1326         assert(kp->ks_data_size == sizeof (short));

1328         (void) asprintf(&value, "%hu", *sh);
1329         SAVE_STRING_X(ksi, "override", value);
1330         free(value);
1331 }
1332 #endif

1334 #ifdef __sparc
1335 static void
1336 save_ps_shadow(kstat_t *kp, ks_instance_t *ksi)
1337 {
1338         uchar_t *uchar = (uchar_t *)(kp->ks_data);

1340         assert(kp->ks_data_size == SYS_PS_COUNT);

1342         SAVE_CHAR_X(ksi, "core_0", *uchar++);
1343         SAVE_CHAR_X(ksi, "core_1", *uchar++);
1344         SAVE_CHAR_X(ksi, "core_2", *uchar++);
1345         SAVE_CHAR_X(ksi, "core_3", *uchar++);
1346         SAVE_CHAR_X(ksi, "core_4", *uchar++);
1347         SAVE_CHAR_X(ksi, "core_5", *uchar++);
1348         SAVE_CHAR_X(ksi, "core_6", *uchar++);
1349         SAVE_CHAR_X(ksi, "core_7", *uchar++);
1350         SAVE_CHAR_X(ksi, "pps_0", *uchar++);
1351         SAVE_CHAR_X(ksi, "clk_33", *uchar++);
1352         SAVE_CHAR_X(ksi, "clk_50", *uchar++);
1353         SAVE_CHAR_X(ksi, "v5_p", *uchar++);
1354         SAVE_CHAR_X(ksi, "v12_p", *uchar++);
1355         SAVE_CHAR_X(ksi, "v5_aux", *uchar++);
1356         SAVE_CHAR_X(ksi, "v5_p_pch", *uchar++);
1357         SAVE_CHAR_X(ksi, "v12_p_pch", *uchar++);
1358         SAVE_CHAR_X(ksi, "v3_pch", *uchar++);
1359         SAVE_CHAR_X(ksi, "v5_pch", *uchar++);
1360         SAVE_CHAR_X(ksi, "p_fan", *uchar++);
1361 }
1362 #endif

1364 #ifdef __sparc
1365 static void
1366 save_fault_list(kstat_t *kp, ks_instance_t *ksi)
1367 {
1368         struct ft_list *fault;
1369         char    name[KSTAT_STRLEN + 7];
1370         int     i;

1372         for (i = 1, fault = (struct ft_list *)(kp->ks_data);
1373             i <= 999999 && i <= kp->ks_data_size / sizeof (struct ft_list);
1374             i++, fault++) {
1375                 (void) snprintf(name, sizeof (name), "unit_%d", i);
1376                 SAVE_INT32_X(ksi, name, fault->unit);
1377                 (void) snprintf(name, sizeof (name), "type_%d", i);
1378                 SAVE_INT32_X(ksi, name, fault->type);
1379                 (void) snprintf(name, sizeof (name), "fclass_%d", i);
```



```
1380                 SAVE_INT32_X(ksi, name, fault->fclass);
1381                 (void) snprintf(name, sizeof (name), "create_time_%d", i);
1382                 SAVE_HRTIME_X(ksi, name, fault->create_time);
1383                 (void) snprintf(name, sizeof (name), "msg_%d", i);
1384                 SAVE_STRING_X(ksi, name, faultp->msg);
1385         }
1386 }
1387 #endif

1389 static void
1390 save_named(kstat_t *kp, ks_instance_t *ksi)
1391 {
1392         kstat_named_t *knp;
1393         int     n;

1395         for (n = kp->ks_ndata, knp = KSTAT_NAMED_PTR(kp); n > 0; n--, knp++) {
1396                 switch (knp->data_type) {
1397                 case KSTAT_DATA_CHAR:
1398                         nvpair_insert(ksi, knp->name,
1399                             (ks_value_t *)&knp->value, KSTAT_DATA_CHAR);
1400                         break;
1401                 case KSTAT_DATA_INT32:
1402                         nvpair_insert(ksi, knp->name,
1403                             (ks_value_t *)&knp->value, KSTAT_DATA_INT32);
1404                         break;
1405                 case KSTAT_DATA_UINT32:
1406                         nvpair_insert(ksi, knp->name,
1407                             (ks_value_t *)&knp->value, KSTAT_DATA_UINT32);
1408                         break;
1409                 case KSTAT_DATA_INT64:
1410                         nvpair_insert(ksi, knp->name,
1411                             (ks_value_t *)&knp->value, KSTAT_DATA_INT64);
1412                         break;
1413                 case KSTAT_DATA_UINT64:
1414                         nvpair_insert(ksi, knp->name,
1415                             (ks_value_t *)&knp->value, KSTAT_DATA_UINT64);
1416                         break;
1417                 case KSTAT_DATA_STRING:
1418                         SAVE_STRING_X(ksi, knp->name, KSTAT_NAMED_STR_PTR(knp));
1419                         break;
1420                 default:
1421                         assert(B_FALSE); /* Invalid data type */
1422                         break;
1423                 }
1424         }
1425 }

1427 static void
1428 save_intr(kstat_t *kp, ks_instance_t *ksi)
1429 {
1430         kstat_intr_t *intr = KSTAT_INTR_PTR(kp);
1431         char    *intr_names[] = {"hard", "soft", "watchdog", "spurious",
1432             "multiple_service"};
1433         int     n;

1435         for (n = 0; n < KSTAT_NUM_INTRS; n++)
1436                 SAVE_UINT32_X(ksi, intr_names[n], intr->intrs[n]);
1437 }

1439 static void
1440 save_io(kstat_t *kp, ks_instance_t *ksi)
1441 {
1442         kstat_io_t      *ksio = KSTAT_IO_PTR(kp);

1444         SAVE_UINT64(ksi, ksio, nread);
1445         SAVE_UINT64(ksi, ksio, nwritten);
```

```
1446         SAVE_UINT32(ksi, ksio, reads);
1447         SAVE_UINT32(ksi, ksio, writes);
1448         SAVE_HRTIME(ksi, ksio, wtime);
1449         SAVE_HRTIME(ksi, ksio, wlentime);
1450         SAVE_HRTIME(ksi, ksio, wlastupdate);
1451         SAVE_HRTIME(ksi, ksio, rtime);
1452         SAVE_HRTIME(ksi, ksio, rlentime);
1453         SAVE_HRTIME(ksi, ksio, rlastupdate);
1454         SAVE_UINT32(ksi, ksio, wcnt);
1455         SAVE_UINT32(ksi, ksio, rcnt);
1456 }

1458 static void
1459 save_timer(kstat_t *kp, ks_instance_t *ksi)
1460 {
1461         kstat_timer_t   *ktimer = KSTAT_TIMER_PTR(kp);

1463         SAVE_STRING(ksi, ktimer, name);
1464         SAVE_UINT64(ksi, ktimer, num_events);
1465         SAVE_HRTIME(ksi, ktimer, elapsed_time);
1466         SAVE_HRTIME(ksi, ktimer, min_time);
1467         SAVE_HRTIME(ksi, ktimer, max_time);
1468         SAVE_HRTIME(ksi, ktimer, start_time);
1469         SAVE_HRTIME(ksi, ktimer, stop_time);
1470 }
1471 #endif /* ! codereview */
```

```
**********************************************************
    7042 Fri Jan 11 07:50:23 2013
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 2013 David Hoeppner.  All rights reserved.
 * Copyright 2013 Nexenta Systems, Inc.  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 <regex.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);      \
```

```
126 }

128 #define SAVE_UINT32_X(I, N, V)                          \
129 {                                                       \
130         ks_value_t v;                                   \
131         v.ui32 = V;                                     \
132         nvpair_insert(I, N, &v, KSTAT_DATA_UINT32);     \
133 }

135 #define SAVE_UINT64_X(I, N, V)                          \
136 {                                                       \
137         ks_value_t v;                                   \
138         v.ui64 = V;                                     \
139         nvpair_insert(I, N, &v, KSTAT_DATA_UINT64);     \
140 }

142 #define SAVE_STRING_X(I, N, V)                          \
143 {                                                       \
144         ks_value_t v;                                   \
145         v.str.addr.ptr = safe_strdup(V);                \
146         v.str.len = strlen(V);                          \
147         nvpair_insert(I, N, &v, KSTAT_DATA_STRING);     \
148 }

150 #define SAVE_CHAR_X(I, N, V)                            \
151 {                                                       \
152         ks_value_t v;                                   \
153         asprintf(&v.str.addr.ptr, "%c", V);             \
154         v.str.len = 1;                                  \
155         nvpair_insert(I, N, &v, KSTAT_DATA_STRING);     \
156 }

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

162 #define JSON_FMT                                        \
163         "{\n\t\"module\": \"%s\",\n"                    \
164         "\t\"instance\": %d,\n"                         \
165         "\t\"name\": \"%s\",\n"                         \
166         "\t\"class\": \"%s\",\n"                        \
167         "\t\"type\": %d,\n"

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

173 typedef struct ks_instance {
174         list_node_t     ks_next;
175         char            ks_name[KSTAT_STRLEN];
176         char            ks_module[KSTAT_STRLEN];
177         char            ks_class[KSTAT_STRLEN];
178         int             ks_instance;
179         uchar_t         ks_type;
180         hrtime_t        ks_snaptime;
181         list_t          ks_nvlist;
182 } ks_instance_t;

184 typedef struct ks_nvpair {
185         list_node_t     nv_next;
186         char            name[KSTAT_STRLEN];
187         uchar_t         data_type;
188         ks_value_t      value;
189 } ks_nvpair_t;

191 typedef struct ks_pattern {
```



```
192         char            *pstr;
193         regex_t         preg;
194 } ks_pattern_t;

196 typedef struct ks_selector {
197         list_node_t     ks_next;
198         ks_pattern_t    ks_module;
199         ks_pattern_t    ks_instance;
200         ks_pattern_t    ks_name;
201         ks_pattern_t    ks_statistic;
202 } ks_selector_t;

204 static void     usage(void);
205 static int      compare_instances(ks_instance_t *, ks_instance_t *);
206 static void     nvpair_insert(ks_instance_t *, char *, ks_value_t *, uchar_t);
207 static boolean_t        ks_match(const char *, ks_pattern_t *);
208 static ks_selector_t    *new_selector(void);
209 static void     ks_instances_read(kstat_ctl_t *);
210 static void     ks_value_print(ks_nvpair_t *);
211 static void     ks_instance_print(ks_instance_t *, ks_nvpair_t *);
212 static void     ks_instances_print(void);
213 static char     *ks_safe_strdup(char *);
214 static void     ks_sleep_until(hrtime_t *, hrtime_t, int, int *);

216 /* Raw kstat readers */
217 static void     save_cpu_stat(kstat_t *, ks_instance_t *);
218 static void     save_var(kstat_t *, ks_instance_t *);
219 static void     save_ncstats(kstat_t *, ks_instance_t *);
220 static void     save_sysinfo(kstat_t *, ks_instance_t *);
221 static void     save_vminfo(kstat_t *, ks_instance_t *);
222 static void     save_nfs(kstat_t *, ks_instance_t *);
223 #ifdef __sparc
224 static void     save_sfmmu_global_stat(kstat_t *, ks_instance_t *);
225 static void     save_sfmmu_tsbsize_stat(kstat_t *, ks_instance_t *);
226 static void     save_simmstat(kstat_t *, ks_instance_t *);
227 /* Helper function for save_temperature() */
228 static char     *short_array_to_string(short *, int);
229 static void     save_temperature(kstat_t *, ks_instance_t *);
230 static void     save_temp_over(kstat_t *, ks_instance_t *);
231 static void     save_ps_shadow(kstat_t *, ks_instance_t *);
232 static void     save_fault_list(kstat_t *, ks_instance_t *);
233 #endif

235 /* Named kstat readers */
236 static void     save_named(kstat_t *, ks_instance_t *);
237 static void     save_intr(kstat_t *, ks_instance_t *);
238 static void     save_io(kstat_t *, ks_instance_t *);
239 static void     save_timer(kstat_t *, ks_instance_t *);

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

244 static struct {
245         kstat_raw_reader_t fn;
246         char *name;
247 } ks_raw_lookup[] = {
248         /* Function name                kstat name              */
249         {save_cpu_stat,                 "cpu_stat:cpu_stat"},
250         {save_var,                      "unix:var"},
251         {save_ncstats,                  "unix:ncstats"},
252         {save_sysinfo,                  "unix:sysinfo"},
253         {save_vminfo,                   "unix:vminfo"},
254         {save_nfs,                      "nfs:mntinfo"},
255 #ifdef __sparc
256         {save_sfmmu_global_stat,        "unix:sfmmu_global_stat"},
257         {save_sfmmu_tsbsize_stat,       "unix:sfmmu_tsbsize_stat"},
```

```
258         {save_simmstat,                 "unix:simm-status"},
259         {save_temperature,              "unix:temperature"},
260         {save_temp_over,                "unix:temperature override"},
261         {save_ps_shadow,                "unix:ps_shadow"},
262         {save_fault_list,               "unix:fault_list"},
263 #endif
264         {NULL, NULL},
265 };

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

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

```
**********************************************************
    9133 Fri Jan 11 07:50:23 2013
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 "Jan 9, 2013"
.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, or a regular expression
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\fR
.ad
.RS 16n
Displays output in parseable format with a colon as separator.
.RE

.sp
.ne 2
.na
#endif /* ! codereview */
\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 or a regular expression enclosed by '/'
 67 may be a shell glob pattern or a Perl regular expression enclosed by '/'
185 characters. It is possible to use both specifier types within a single operand.
186 Leaving a specifier empty is equivalent to using the '*' glob pattern for that
```



```
187 specifier.
188 .RE

190 .sp
191 .ne 2
192 .na
193 \fB\fIinterval\fR\fR
194 .ad
195 .sp .6
196 .RS 4n
197 The number of seconds between reports.
198 .RE

200 .sp
201 .ne 2
202 .na
203 \fB\fIcount\fR\fR
204 .ad
205 .sp .6
206 .RS 4n
207 The number of reports to be printed.
208 .RE

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

227 unix:0:system_misc:avenrun_15min        3
228 unix:0:system_misc:avenrun_1min 4
229 unix:0:system_misc:avenrun_5min 2
230 .fi
231 .in -2
232 .sp

234 .LP
235 \fBExample 2 \fRUsing the \fBkstat\fR Command
236 .sp
237 .in +2
238 .nf
239 example$ \fBkstat -p -m cpu_stat -s 'intr*'\fR
240 example$ \fBkstat -p cpu_stat:::/^intr/\fR

242 cpu_stat:0:cpu_stat0:intr       29682330
243 cpu_stat:0:cpu_stat0:intrblk    87
244 cpu_stat:0:cpu_stat0:intrthread 15054222
245 cpu_stat:1:cpu_stat1:intr       426073
246 cpu_stat:1:cpu_stat1:intrblk    51
247 cpu_stat:1:cpu_stat1:intrthread 289668
248 cpu_stat:2:cpu_stat2:intr       134160
249 cpu_stat:2:cpu_stat2:intrblk    0
250 cpu_stat:2:cpu_stat2:intrthread 131
251 cpu_stat:3:cpu_stat3:intr       196566
252 cpu_stat:3:cpu_stat3:intrblk    30
```

```
253 cpu_stat:3:cpu_stat3:intrthread 59626
254 .fi
255 .in -2
256 .sp

258 .LP
259 \fBExample 3 \fRUsing the \fBkstat\fR Command
260 .sp
261 .in +2
262 .nf
263 example$ \fBkstat -p :::state ':::avenrun*'\fR
264 example$ \fBkstat -p :::state :::/^avenrun/\fR

266 cpu_info:0:cpu_info0:state      on-line
267 cpu_info:1:cpu_info1:state      on-line
268 cpu_info:2:cpu_info2:state      on-line
269 cpu_info:3:cpu_info3:state      on-line
270 unix:0:system_misc:avenrun_15min        4
271 unix:0:system_misc:avenrun_1min 10
272 unix:0:system_misc:avenrun_5min 3
273 .fi
274 .in -2
275 .sp

277 .LP
278 \fBExample 4 \fRUsing the \fBkstat\fR Command
279 .sp
280 .in +2
281 .nf
282 example$ \fBkstat -p 'unix:0:system_misc:avenrun*' 1 3\fR
283 unix:0:system_misc:avenrun_15min        15
284 unix:0:system_misc:avenrun_1min 11
285 unix:0:system_misc:avenrun_5min 21

287 unix:0:system_misc:avenrun_15min        15
288 unix:0:system_misc:avenrun_1min 11
289 unix:0:system_misc:avenrun_5min 21

291 unix:0:system_misc:avenrun_15min        15
292 unix:0:system_misc:avenrun_1min 11
293 unix:0:system_misc:avenrun_5min 21
294 .fi
295 .in -2
296 .sp

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

309 Thu Jul 22 19:39:55 1999
310 unix:0:system_misc:avenrun_15min        12
311 unix:0:system_misc:avenrun_1min 0
312 unix:0:system_misc:avenrun_5min 11
313 .fi
314 .in -2
315 .sp

317 .LP
318 \fBExample 6 \fRUsing the \fBkstat\fR Command
```



```
319 .sp
320 .in +2
321 .nf
322 example$ \fBkstat -p -T u 'unix:0:system_misc:avenrun*'\fR
323 932668656
324 unix:0:system_misc:avenrun_15min        14
325 unix:0:system_misc:avenrun_1min 5
326 unix:0:system_misc:avenrun_5min 18
327 .fi
328 .in -2
329 .sp

331 .SH EXIT STATUS
332 .sp
333 .LP
334 The following exit values are returned:
335 .sp
336 .ne 2
337 .na
338 \fB\fB0\fR\fR
339 .ad
340 .RS 5n
341 One or more statistics were matched.
342 .RE

344 .sp
345 .ne 2
346 .na
347 \fB\fB1\fR\fR
348 .ad
349 .RS 5n
350 No statistics were matched.
351 .RE

353 .sp
354 .ne 2
355 .na
356 \fB\fB2\fR\fR
357 .ad
358 .RS 5n
359 Invalid command line options were specified.
360 .RE

362 .sp
363 .ne 2
364 .na
365 \fB\fB3\fR\fR
366 .ad
367 .RS 5n
368 A fatal error occurred.
369 .RE

371 .SH FILES
372 .sp
373 .ne 2
374 .na
375 \fB\fB/dev/kstat\fR\fR
376 .ad
377 .RS 14n
378 kernel statistics driver
379 .RE

381 .SH SEE ALSO
382 .sp
383 .LP
384 \fBdate\fR(1), \fBsh\fR(1), \fBtime\fR(2), \fBgmatch\fR(3GEN),
```

```
385 \fBkstat\fR(3KSTAT), \fBattributes\fR(5), \fBkstat\fR(7D), \fBsd\fR(7D),
386 \fBkstat\fR(9S)
387 .SH NOTES
388 .sp
389 .LP
390 If the pattern argument contains glob or RE metacharacters which are also
273 If the pattern argument contains glob or Perl RE metacharacters which are also
391 shell metacharacters, it will be necessary to enclose the pattern with
392 appropriate shell quotes.
```