```
**********************************************************
    1267 Thu Aug 30 18:01:15 2012
new/usr/src/cmd/stat/Makefile
749 "/usr/bin/kstat" should be rewritten in C
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #
  21 #
  22 # Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  23 # Use is subject to license terms.
  24 #
  25 #ident  "%Z%%M% %I%     %E% SMI"
  26 #
  25 # cmd/stat/Makefile
  26 #

  28 include ../Makefile.cmd

  30 SUBDIRS=        iostat mpstat vmstat fsstat kstat
  32 SUBDIRS=        iostat mpstat vmstat fsstat

  32 all :=          TARGET = all
  33 install :=      TARGET = install
  34 clean :=        TARGET = clean
  35 clobber :=      TARGET = clobber
  36 lint :=         TARGET = lint
  37 _msg :=         TARGET = _msg

  39 .KEEP_STATE:

  41 all install lint clean clobber _msg: $(SUBDIRS)

  43 $(SUBDIRS): FRC
  44         @cd $@; pwd; $(MAKE) $(MFLAGS) $(TARGET)

  46 FRC:
```

****
1511 Thu Aug 30 18:01:15 2012
new/usr/src/cmd/stat/kstat/Makefile
749 "/usr/bin/kstat" should be rewritten in C
****

```
#
# 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}
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 */
```

```
**********************************************************
   33403 Thu Aug 30 18:01:16 2012
new/usr/src/cmd/stat/kstat/kstat.c
749 "/usr/bin/kstat" should be rewritten in C
**********************************************************
```

```
/*
 * 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 <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";
int     caught_cont = 0;

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.
         */
        (void) list_create(&selector_list, sizeof (ks_selector_t),
            offsetof(ks_selector_t, ks_next));

        nselector = new_selector();

        /*
         * Parse named command line arguments.
         */
        while ((c = getopt(argc, argv, "h?CqjlpT:m:i:n:s:c:")) != EOF)
                switch (c) {
                case 'h':
```

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

188         if (g_qflg && (g_jflg || g_pflg)) {
189                 (void) fprintf(stderr, gettext(
190                     "-q and -lpj are mutually exclusive\n"));
191                 errflg = B_TRUE;
192         }
```



```
194         if (errflg) {
195                 usage();
196                 exit(2);
197         }

199         argc -= optind;
200         argv += optind;

202         /*
203          * Consume the rest of the command line. Parsing the
204          * unnamed command line arguments.
205          */
206         while (argc--) {
207                 errno = 0;
208                 tmp = strtoul(*argv, &q, 10);
209                 if (tmp == ULONG_MAX && errno == ERANGE) {
210                         if (n == 0) {
211                                 (void) fprintf(stderr, gettext(
212                                     "Interval is too large\n"));
213                         } else if (n == 1) {
214                                 (void) fprintf(stderr, gettext(
215                                     "Count is too large\n"));
216                         }
217                         usage();
218                         exit(2);
219                 }

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

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

256                         if (m < 4) {
257                                 free(uselector);
258                                 usage();
259                                 exit(2);
```

```
260                         }
262                         uselflg = B_TRUE;
263                         list_insert_tail(&selector_list, uselector);
264                 } else {
265                         if (tmp < 1) {
266                                 if (n == 0) {
267                                         (void) fprintf(stderr, gettext(
268                                             "Interval must be an "
269                                             "integer >= 1"));
270                                 } else if (n == 1) {
271                                         (void) fprintf(stderr, gettext(
272                                             "Count must be an integer >= 1"));
273                                 }
274                                 usage();
275                                 exit(2);
276                         } else {
277                                 if (n == 0) {
278                                         interval = tmp;
279                                         count = -1;
280                                 } else if (n == 1) {
281                                         count = tmp;
282                                 } else {
283                                         usage();
284                                         exit(2);
285                                 }
286                         }
287                         n++;
288                 }
289                 argv++;
290         }
292         /*
293          * Check if we founded a named selector on the cmdline.
294          */
295         if (uselflg) {
296                 if (nselflg) {
297                         (void) fprintf(stderr, gettext(
298                             "module:instance:name:statistic and "
299                             "-m -i -n -s are mutually exclusive"));
300                         usage();
301                         exit(2);
302                 } else {
303                         free(nselector);
304                 }
305         } else {
306                 list_insert_tail(&selector_list, nselector);
307         }
309         assert(!list_is_empty(&selector_list));
311         (void) list_create(&instances_list, sizeof (ks_instance_t),
312             offsetof(ks_instance_t, ks_next));
314         kc = kstat_open();
315         if (kc == NULL)  {
316                 perror("kstat_open");
317                 exit(3);
318         }
320         period_n = (hrtime_t)interval * NANOSEC;
321         start_n = gethrtime();
323         while (count == -1 || count-- > 0) {
324                 ks_instances_read(kc);
325                 ks_instances_print();
```



```
327                 if (interval && count) {
328                         sleep_until(&start_n, period_n, infinite_cycles,
329                             &caught_cont);
330                         (void) kstat_chain_update(kc);
331                         (void) putchar('\n');
332                 }
333         }
335         (void) kstat_close(kc);
337         return (g_matched);
338 }
340 /*
341  * Print usage.
342  */
343 static void
344 usage(void)
345 {
346         (void) fprintf(stderr, gettext(
347             "Usage:\n"
348             "kstat [ -Cjlpq ] [ -T d|u ] [ -c class ]\n"
349             "      [ -m module ] [ -i instance ] [ -n name ] [ -s statistic ]\n"
350             "      [ interval [ count ] ]\n"
351             "kstat [ -Cjlpq ] [ -T d|u ] [ -c class ]\n"
352             "      [ module:instance:name:statistic ... ]\n"
353             "      [ interval [ count ] ]\n"));
354 }
356 /*
357  * Sort compare function.
358  */
359 static int
360 compare_instances(ks_instance_t *l_arg, ks_instance_t *r_arg)
361 {
362         int     rval;
364         rval = strcasecmp(l_arg->ks_module, r_arg->ks_module);
365         if (rval == 0) {
366                 if (l_arg->ks_instance == r_arg->ks_instance) {
367                         return (strcasecmp(l_arg->ks_name, r_arg->ks_name));
368                 } else if (l_arg->ks_instance < r_arg->ks_instance) {
369                         return (-1);
370                 } else {
371                         return (1);
372                 }
373         } else {
374                 return (rval);
375         }
376 }
378 /*
379  * Inserts an instance in the per selector list.
380  */
381 static void
382 nvpair_insert(ks_instance_t *ksi, char *name, ks_value_t *value,
383     uchar_t data_type)
384 {
385         ks_nvpair_t     *instance;
386         ks_nvpair_t     *tmp;
388         instance = (ks_nvpair_t *)malloc(sizeof (ks_nvpair_t));
389         if (instance == NULL) {
390                 perror("malloc");
391                 exit(3);
```

```
392         }

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

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

402         list_insert_before(&ksi->ks_nvlist, tmp, instance);
403 }

405 /*
406  * Allocates a new all-matching selector.
407  */
408 static ks_selector_t *
409 new_selector(void)
410 {
411         ks_selector_t   *selector;

413         selector = (ks_selector_t *)malloc(sizeof (ks_selector_t));
414         if (selector == NULL) {
415                 perror("malloc");
416                 exit(3);
417         }

419         (void) list_link_init(&selector->ks_next);

421         selector->ks_module = "*";
422         selector->ks_instance = "*";
423         selector->ks_name = "*";
424         selector->ks_statistic = "*";

426         return (selector);
427 }

429 /*
430  * This function was taken from the perl kstat module code - please
431  * see for further comments there.
432  */
433 static kstat_raw_reader_t
434 lookup_raw_kstat_fn(char *module, char *name)
435 {
436         char            key[KSTAT_STRLEN * 2];
437         register char   *f, *t;
438         int             n = 0;

440         for (f = module, t = key; *f != '\0'; f++, t++) {
441                 while (*f != '\0' && isdigit(*f))
442                         f++;
443                 *t = *f;
444         }
445         *t++ = ':';

447         for (f = name; *f != '\0'; f++, t++) {
448                 while (*f != '\0' && isdigit(*f))
449                         f++;
450                 *t = *f;
451         }
452         *t = '\0';

454         while (ks_raw_lookup[n].fn != NULL) {
455                 if (strncmp(ks_raw_lookup[n].name, key, strlen(key)) == 0)
456                         return (ks_raw_lookup[n].fn);
457                 n++;
```



```
458         }

460         return (0);
461 }

463 /*
464  * Iterate over all kernel statistics and save matches.
465  */
466 static void
467 ks_instances_read(kstat_ctl_t *kc)
468 {
469         kstat_raw_reader_t save_raw;
470         kid_t           id;
471         ks_selector_t   *selector;
472         ks_instance_t   *ksi;
473         ks_instance_t   *tmp;
474         kstat_t         *kp;
475         boolean_t       skip;
476         char            *ks_number;

478         for (kp = kc->kc_chain; kp != NULL; kp = kp->ks_next) {
479                 /* Don't bother storing the kstat headers */
480                 if (strncmp(kp->ks_name, "kstat_", 6) == 0) {
481                         continue;
482                 }

484                 /* Don't bother storing raw stats we don't understand */
485                 if (kp->ks_type == KSTAT_TYPE_RAW) {
486                         save_raw = lookup_raw_kstat_fn(kp->ks_module,
487                             kp->ks_name);
488                         if (save_raw == NULL) {
489 #ifdef REPORT_UNKNOWN
490                                 (void) fprintf(stderr,
491                                     "Unknown kstat type %s:%d:%s - "
492                                     "%d of size %d\n", kp->ks_module,
493                                     kp->ks_instance, kp->ks_name,
494                                     kp->ks_ndata, kp->ks_data_size);
495 #endif
496                                 continue;
497                         }
498                 }

500                 /*
501                  * Iterate over the list of selectors and skip
502                  * instances we dont want. We filter for statistics
503                  * later, as we dont know them yet.
504                  */
505                 skip = B_FALSE;
506                 (void) asprintf(&ks_number, "%d", kp->ks_instance);
507                 selector = list_head(&selector_list);
508                 while (selector != NULL) {
509                         if (!(gmatch(kp->ks_module, selector->ks_module) != 0 &&
510                             gmatch(ks_number, selector->ks_instance) != 0 &&
511                             gmatch(kp->ks_name, selector->ks_name) != 0 &&
512                             gmatch(kp->ks_class, g_ks_class))) {
513                                 skip = B_TRUE;
514                         }
515                         selector = list_next(&selector_list, selector);
516                 }

518                 free(ks_number);

520                 if (skip) {
521                         continue;
522                 }
```

```
524                 /*
525                  * Allocate a new instance and fill in the values
526                  * we know so far.
527                  */
528                 ksi = (ks_instance_t *)malloc(sizeof (ks_instance_t));
529                 if (ksi == NULL) {
530                         perror("malloc");
531                         exit(3);
532                 }

534                 (void) list_link_init(&ksi->ks_next);

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

540                 ksi->ks_instance = kp->ks_instance;

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

545                 SAVE_HRTIME_X(ksi, "crtime", kp->ks_crtime);
546                 SAVE_HRTIME_X(ksi, "snaptime", kp->ks_snaptime);
547                 if (g_pflg) {
548                         SAVE_STRING_X(ksi, "class", kp->ks_class);
549                 }

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

556                 list_insert_before(&instances_list, tmp, ksi);

558                 /* Read the actual statistics */
559                 id = kstat_read(kc, kp, NULL);
560                 if (id == -1) {
561                         perror("kstat_read");
562                         continue;
563                 }

565                 switch (kp->ks_type) {
566                 case KSTAT_TYPE_RAW:
567                         save_raw(kp, ksi);
568                         break;
569                 case KSTAT_TYPE_NAMED:
570                         save_named(kp, ksi);
571                         break;
572                 case KSTAT_TYPE_INTR:
573                         save_intr(kp, ksi);
574                         break;
575                 case KSTAT_TYPE_IO:
576                         save_io(kp, ksi);
577                         break;
578                 case KSTAT_TYPE_TIMER:
579                         save_timer(kp, ksi);
580                         break;
581                 default:
582                         assert(B_FALSE); /* Invalid type */
583                         break;
584                 }
585         }
586 }

588 /*
589  * Print the value of a name-value pair.
```



```
590  */
591 static void
592 ks_value_print(ks_nvpair_t *nvpair)
593 {
594         switch (nvpair->data_type) {
595         case KSTAT_DATA_CHAR:
596                 (void) fprintf(stdout, "%s", nvpair->value.c);
597                 break;
598         case KSTAT_DATA_INT32:
599                 (void) fprintf(stdout, "%d", nvpair->value.i32);
600                 break;
601         case KSTAT_DATA_UINT32:
602                 (void) fprintf(stdout, "%u", nvpair->value.ui32);
603                 break;
604         case KSTAT_DATA_INT64:
605                 (void) fprintf(stdout, "%lld", nvpair->value.i64);
606                 break;
607         case KSTAT_DATA_UINT64:
608                 (void) fprintf(stdout, "%llu", nvpair->value.ui64);
609                 break;
610         case KSTAT_DATA_STRING:
611                 (void) fprintf(stdout, "%s", KSTAT_NAMED_STR_PTR(nvpair));
612                 break;
613         case KSTAT_DATA_HRTIME:
614                 if (nvpair->value.ui64 == 0)
615                         (void) fprintf(stdout, "0");
616                 else
617                         (void) fprintf(stdout, "%.9f",
618                             nvpair->value.ui64 / 1000000000.0);
619                 break;
620         default:
621                 assert(B_FALSE);
622         }
623 }

625 /*
626  * Print a single instance.
627  */
628 static void
629 ks_instance_print(ks_instance_t *ksi, ks_nvpair_t *nvpair)
630 {
631         if (g_headerflg) {
632                 if (g_jflg) {
633                         (void) fprintf(stdout, JSON_FMT,
634                             ksi->ks_module, ksi->ks_instance,
635                             ksi->ks_name, ksi->ks_class);
636                 } else if (!g_pflg) {
637                         (void) fprintf(stdout, DFLT_FMT,
638                             ksi->ks_module, ksi->ks_instance,
639                             ksi->ks_name, ksi->ks_class);
640                 }
641                 g_headerflg = B_FALSE;
642         }

644         if (g_jflg) {
645                 (void) fprintf(stdout, KS_JFMT, nvpair->name);
646                 if (nvpair->data_type == KSTAT_DATA_STRING) {
647                         (void) putchar('\"');
648                         ks_value_print(nvpair);
649                         (void) putchar('\"');
650                 } else {
651                         ks_value_print(nvpair);
652                 }
653                 if (nvpair != list_tail(&ksi->ks_nvlist))
654                         (void) putchar(',');
655         } else if (g_pflg) {
```

```
656                 (void) fprintf(stdout, KS_PFMT,
657                     ksi->ks_module, ksi->ks_instance,
658                     ksi->ks_name, nvpair->name);
659                 if (!g_lflg) {
660                         (void) putchar(g_cflg ? ':': '\t');
661                         ks_value_print(nvpair);
662                 }
663         } else {
664                 (void) fprintf(stdout, KS_DFMT, nvpair->name);
665                 ks_value_print(nvpair);
666         }

668         (void) putchar('\n');
669 }

671 /*
672  * Print all instances.
673  */
674 static void
675 ks_instances_print(void)
676 {
677         ks_selector_t   *selector;
678         ks_instance_t   *ksi, *ktmp;
679         ks_nvpair_t     *nvpair, *ntmp;

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

684         if (g_jflg)
685                 (void) putchar('[');

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

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

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

704                                 g_matched = 0;
705                                 if (!g_qflg) {
706                                         ks_instance_print(ksi, nvpair);
707                                 }
708                         }

710                         if (!g_headerflg) {
711                                 if (g_jflg) {
712                                         (void) fprintf(stdout, "\t}\n}");
713                                         if (ksi != list_tail(&instances_list))
714                                                 (void) putchar(',');
715                                 } else if (!g_pflg) {
716                                         (void) putchar('\n');
717                                 }
718                         }
719                 }

721                 selector = list_next(&selector_list, selector);
```



```
722         }

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

727         (void) fflush(stdout);

729         /* Free the instances list */
730         ksi = list_head(&instances_list);
731         while (ksi != NULL) {
732                 nvpair = list_head(&ksi->ks_nvlist);
733                 while (nvpair != NULL) {
734                         ntmp = nvpair;
735                         nvpair = list_next(&ksi->ks_nvlist, nvpair);
736                         list_remove(&ksi->ks_nvlist, ntmp);
737                         if (ntmp->data_type == KSTAT_DATA_STRING)
738                                 free(ntmp->value.str.addr.ptr);
739                         free(ntmp);
740                 }

742                 ktmp = ksi;
743                 ksi = list_next(&instances_list, ksi);
744                 list_remove(&instances_list, ktmp);
745                 list_destroy(&ktmp->ks_nvlist);
746                 free(ktmp);
747         }
748 }

750 static void
751 save_cpu_stat(kstat_t *kp, ks_instance_t *ksi)
752 {
753         cpu_stat_t      *stat;
754         cpu_sysinfo_t   *sysinfo;
755         cpu_syswait_t   *syswait;
756         cpu_vminfo_t    *vminfo;

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

763         SAVE_UINT32_X(ksi, "idle", sysinfo->cpu[CPU_IDLE]);
764         SAVE_UINT32_X(ksi, "user", sysinfo->cpu[CPU_USER]);
765         SAVE_UINT32_X(ksi, "kernel", sysinfo->cpu[CPU_KERNEL]);
766         SAVE_UINT32_X(ksi, "wait", sysinfo->cpu[CPU_WAIT]);
767         SAVE_UINT32_X(ksi, "wait_io", sysinfo->cpu[W_IO]);
768         SAVE_UINT32_X(ksi, "wait_swap", sysinfo->cpu[W_SWAP]);
769         SAVE_UINT32_X(ksi, "wait_pio", sysinfo->cpu[W_PIO]);
770         SAVE_UINT32(ksi, sysinfo, bread);
771         SAVE_UINT32(ksi, sysinfo, bwrite);
772         SAVE_UINT32(ksi, sysinfo, lread);
773         SAVE_UINT32(ksi, sysinfo, lwrite);
774         SAVE_UINT32(ksi, sysinfo, phread);
775         SAVE_UINT32(ksi, sysinfo, phwrite);
776         SAVE_UINT32(ksi, sysinfo, pswitch);
777         SAVE_UINT32(ksi, sysinfo, trap);
778         SAVE_UINT32(ksi, sysinfo, intr);
779         SAVE_UINT32(ksi, sysinfo, syscall);
780         SAVE_UINT32(ksi, sysinfo, sysread);
781         SAVE_UINT32(ksi, sysinfo, syswrite);
782         SAVE_UINT32(ksi, sysinfo, sysfork);
783         SAVE_UINT32(ksi, sysinfo, sysvfork);
784         SAVE_UINT32(ksi, sysinfo, sysexec);
785         SAVE_UINT32(ksi, sysinfo, readch);
786         SAVE_UINT32(ksi, sysinfo, writech);
787         SAVE_UINT32(ksi, sysinfo, rcvint);
```

```
788         SAVE_UINT32(ksi, sysinfo, xmtint);
789         SAVE_UINT32(ksi, sysinfo, mdmint);
790         SAVE_UINT32(ksi, sysinfo, rawch);
791         SAVE_UINT32(ksi, sysinfo, canch);
792         SAVE_UINT32(ksi, sysinfo, outch);
793         SAVE_UINT32(ksi, sysinfo, msg);
794         SAVE_UINT32(ksi, sysinfo, sema);
795         SAVE_UINT32(ksi, sysinfo, namei);
796         SAVE_UINT32(ksi, sysinfo, ufsiget);
797         SAVE_UINT32(ksi, sysinfo, ufsdirblk);
798         SAVE_UINT32(ksi, sysinfo, ufsipage);
799         SAVE_UINT32(ksi, sysinfo, ufsinopage);
800         SAVE_UINT32(ksi, sysinfo, inodeovf);
801         SAVE_UINT32(ksi, sysinfo, fileovf);
802         SAVE_UINT32(ksi, sysinfo, procovf);
803         SAVE_UINT32(ksi, sysinfo, intrthread);
804         SAVE_UINT32(ksi, sysinfo, intrblk);
805         SAVE_UINT32(ksi, sysinfo, idlethread);
806         SAVE_UINT32(ksi, sysinfo, inv_swtch);
807         SAVE_UINT32(ksi, sysinfo, nthreads);
808         SAVE_UINT32(ksi, sysinfo, cpumigrate);
809         SAVE_UINT32(ksi, sysinfo, xcalls);
810         SAVE_UINT32(ksi, sysinfo, mutex_adenters);
811         SAVE_UINT32(ksi, sysinfo, rw_rdfails);
812         SAVE_UINT32(ksi, sysinfo, rw_wrfails);
813         SAVE_UINT32(ksi, sysinfo, modload);
814         SAVE_UINT32(ksi, sysinfo, modunload);
815         SAVE_UINT32(ksi, sysinfo, bawrite);
816 #ifdef  STATISTICS      /* see header file */
817         SAVE_UINT32(ksi, sysinfo, rw_enters);
818         SAVE_UINT32(ksi, sysinfo, win_uo_cnt);
819         SAVE_UINT32(ksi, sysinfo, win_uu_cnt);
820         SAVE_UINT32(ksi, sysinfo, win_so_cnt);
821         SAVE_UINT32(ksi, sysinfo, win_su_cnt);
822         SAVE_UINT32(ksi, sysinfo, win_suo_cnt);
823 #endif

825         SAVE_INT32(ksi, syswait, iowait);
826         SAVE_INT32(ksi, syswait, swap);
827         SAVE_INT32(ksi, syswait, physio);

829         SAVE_UINT32(ksi, vminfo, pgrec);
830         SAVE_UINT32(ksi, vminfo, pgfrec);
831         SAVE_UINT32(ksi, vminfo, pgin);
832         SAVE_UINT32(ksi, vminfo, pgpgin);
833         SAVE_UINT32(ksi, vminfo, pgout);
834         SAVE_UINT32(ksi, vminfo, pgpgout);
835         SAVE_UINT32(ksi, vminfo, swapin);
836         SAVE_UINT32(ksi, vminfo, pgswapin);
837         SAVE_UINT32(ksi, vminfo, swapout);
838         SAVE_UINT32(ksi, vminfo, pgswapout);
839         SAVE_UINT32(ksi, vminfo, zfod);
840         SAVE_UINT32(ksi, vminfo, dfree);
841         SAVE_UINT32(ksi, vminfo, scan);
842         SAVE_UINT32(ksi, vminfo, rev);
843         SAVE_UINT32(ksi, vminfo, hat_fault);
844         SAVE_UINT32(ksi, vminfo, as_fault);
845         SAVE_UINT32(ksi, vminfo, maj_fault);
846         SAVE_UINT32(ksi, vminfo, cow_fault);
847         SAVE_UINT32(ksi, vminfo, prot_fault);
848         SAVE_UINT32(ksi, vminfo, softlock);
849         SAVE_UINT32(ksi, vminfo, kernel_asflt);
850         SAVE_UINT32(ksi, vminfo, pgrrun);
851         SAVE_UINT32(ksi, vminfo, execpgin);
852         SAVE_UINT32(ksi, vminfo, execpgout);
853         SAVE_UINT32(ksi, vminfo, execfree);
```



```
854         SAVE_UINT32(ksi, vminfo, anonpgin);
855         SAVE_UINT32(ksi, vminfo, anonpgout);
856         SAVE_UINT32(ksi, vminfo, anonfree);
857         SAVE_UINT32(ksi, vminfo, fspgin);
858         SAVE_UINT32(ksi, vminfo, fspgout);
859         SAVE_UINT32(ksi, vminfo, fsfree);
860 }

862 static void
863 save_var(kstat_t *kp, ks_instance_t *ksi)
864 {
865         struct var      *var = (struct var *)(kp->ks_data);

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

869         SAVE_INT32(ksi, var, v_buf);
870         SAVE_INT32(ksi, var, v_call);
871         SAVE_INT32(ksi, var, v_proc);
872         SAVE_INT32(ksi, var, v_maxupttl);
873         SAVE_INT32(ksi, var, v_nglobpris);
874         SAVE_INT32(ksi, var, v_maxsyspri);
875         SAVE_INT32(ksi, var, v_clist);
876         SAVE_INT32(ksi, var, v_maxup);
877         SAVE_INT32(ksi, var, v_hbuf);
878         SAVE_INT32(ksi, var, v_hmask);
879         SAVE_INT32(ksi, var, v_pbuf);
880         SAVE_INT32(ksi, var, v_sptmap);
881         SAVE_INT32(ksi, var, v_maxpmem);
882         SAVE_INT32(ksi, var, v_autoup);
883         SAVE_INT32(ksi, var, v_bufhwm);
884 }

886 static void
887 save_ncstats(kstat_t *kp, ks_instance_t *ksi)
888 {
889         struct ncstats  *ncstats = (struct ncstats *)(kp->ks_data);

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

893         SAVE_INT32(ksi, ncstats, hits);
894         SAVE_INT32(ksi, ncstats, misses);
895         SAVE_INT32(ksi, ncstats, enters);
896         SAVE_INT32(ksi, ncstats, dbl_enters);
897         SAVE_INT32(ksi, ncstats, long_enter);
898         SAVE_INT32(ksi, ncstats, long_look);
899         SAVE_INT32(ksi, ncstats, move_to_front);
900         SAVE_INT32(ksi, ncstats, purges);
901 }

903 static void
904 save_sysinfo(kstat_t *kp, ks_instance_t *ksi)
905 {
906         sysinfo_t       *sysinfo = (sysinfo_t *)(kp->ks_data);

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

910         SAVE_UINT32(ksi, sysinfo, updates);
911         SAVE_UINT32(ksi, sysinfo, runque);
912         SAVE_UINT32(ksi, sysinfo, runocc);
913         SAVE_UINT32(ksi, sysinfo, swpque);
914         SAVE_UINT32(ksi, sysinfo, swpocc);
915         SAVE_UINT32(ksi, sysinfo, waiting);
916 }

918 static void
919 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);
```

```
1052         SAVE_INT32(ksi, sfmmut, sf_tsbsz_64k);
1053         SAVE_INT32(ksi, sfmmut, sf_tsbsz_128k);
1054         SAVE_INT32(ksi, sfmmut, sf_tsbsz_256k);
1055         SAVE_INT32(ksi, sfmmut, sf_tsbsz_512k);
1056         SAVE_INT32(ksi, sfmmut, sf_tsbsz_1m);
1057         SAVE_INT32(ksi, sfmmut, sf_tsbsz_2m);
1058         SAVE_INT32(ksi, sfmmut, sf_tsbsz_4m);
1059 }
1060 #endif

1062 #ifdef __sparc
1063 static void
1064 save_simmstat(kstat_t *kp, ks_instance_t *ksi)
1065 {
1066         uchar_t *simmstat;
1067         char    *simm_buf;
1068         char    *list = NULL;
1069         int     i;

1071         assert(kp->ks_data_size == sizeof (uchar_t) * SIMM_COUNT);

1073         for (i = 0, simmstat = (uchar_t *)(kp->ks_data); i < SIMM_COUNT - 1;
1074             i++, simmstat++) {
1075                 if (list == NULL) {
1076                         (void) asprintf(&simm_buf, "%d,", *simmstat);
1077                 } else {
1078                         (void) asprintf(&simm_buf, "%s%d,", list, *simmstat);
1079                         free(list);
1080                 }
1081                 list = simm_buf;
1082         }

1084         (void) asprintf(&simm_buf, "%s%d", list, *simmstat);
1085         SAVE_STRING_X(ksi, "status", simm_buf);
1086         free(list);
1087         free(simm_buf);
1088 }
1089 #endif

1091 #ifdef __sparc
1092 /*
1093  * Helper function for save_temperature().
1094  */
1095 static char *
1096 short_array_to_string(short *shortp, int len)
1097 {
1098         char    *list = NULL;
1099         char    *list_buf;

1101         for (; len > 1; len--, shortp++) {
1102                 if (list == NULL) {
1103                         (void) asprintf(&list_buf, "%d,", *shortp);
1104                 } else {
1105                         (void) asprintf(&list_buf, "%s%d,", list, *shortp);
1106                         free(list);
1107                 }
1108                 list = list_buf;
1109         }

1111         (void) asprintf(&list_buf, "%s%s", list, *shortp);
1112         free(list);
1113         return (list_buf);
1114 }

1116 static void
1117 save_temperature(kstat_t *kp, ks_instance_t *ksi)
```



```
1118 {
1119         struct temp_stats *temps = (struct temp_stats *)(kp->ks_data);
1120         char    *buf;
1121         int     n = 1;

1123         assert(kp->ks_data_size == sizeof (struct temp_stats));

1125         SAVE_UINT32(ksi, temps, index);

1127         buf = short_array_to_string(temps->l1, L1_SZ);
1128         SAVE_STRING_X(ksi, "l1", buf);
1129         free(buf);

1131         buf = short_array_to_string(temps->l2, L2_SZ);
1132         SAVE_STRING_X(ksi, "l2", buf);
1133         free(buf);

1135         buf = short_array_to_string(temps->l3, L3_SZ);
1136         SAVE_STRING_X(ksi, "l3", buf);
1137         free(buf);

1139         buf = short_array_to_string(temps->l4, L4_SZ);
1140         SAVE_STRING_X(ksi, "l4", buf);
1141         free(buf);

1143         buf = short_array_to_string(temps->l5, L5_SZ);
1144         SAVE_STRING_X(ksi, "l5", buf);
1145         free(buf);

1147         SAVE_INT32(ksi, temps, max);
1148         SAVE_INT32(ksi, temps, min);
1149         SAVE_INT32(ksi, temps, state);
1150         SAVE_INT32(ksi, temps, temp_cnt);
1151         SAVE_INT32(ksi, temps, shutdown_cnt);
1152         SAVE_INT32(ksi, temps, version);
1153         SAVE_INT32(ksi, temps, trend);
1154         SAVE_INT32(ksi, temps, override);
1155 }
1156 #endif

1158 #ifdef __sparc
1159 static void
1160 save_temp_over(kstat_t *kp, ks_instance_t *ksi)
1161 {
1162         short   *sh = (short *)(kp->ks_data);
1163         char    *value;

1165         assert(kp->ks_data_size == sizeof (short));

1167         (void) asprintf(&value, "%hu", *sh);
1168         SAVE_STRING_X(ksi, "override", value);
1169         free(value);
1170 }
1171 #endif

1173 #ifdef __sparc
1174 static void
1175 save_ps_shadow(kstat_t *kp, ks_instance_t *ksi)
1176 {
1177         uchar_t *uchar = (uchar_t *)(kp->ks_data);

1179         assert(kp->ks_data_size == SYS_PS_COUNT);

1181         SAVE_CHAR_X(ksi, "core_0", *uchar++);
1182         SAVE_CHAR_X(ksi, "core_1", *uchar++);
1183         SAVE_CHAR_X(ksi, "core_2", *uchar++);
```

```
1184         SAVE_CHAR_X(ksi, "core_3", *uchar++);
1185         SAVE_CHAR_X(ksi, "core_4", *uchar++);
1186         SAVE_CHAR_X(ksi, "core_5", *uchar++);
1187         SAVE_CHAR_X(ksi, "core_6", *uchar++);
1188         SAVE_CHAR_X(ksi, "core_7", *uchar++);
1189         SAVE_CHAR_X(ksi, "pps_0", *uchar++);
1190         SAVE_CHAR_X(ksi, "clk_33", *uchar++);
1191         SAVE_CHAR_X(ksi, "clk_50", *uchar++);
1192         SAVE_CHAR_X(ksi, "v5_p", *uchar++);
1193         SAVE_CHAR_X(ksi, "v12_p", *uchar++);
1194         SAVE_CHAR_X(ksi, "v5_aux", *uchar++);
1195         SAVE_CHAR_X(ksi, "v5_p_pch", *uchar++);
1196         SAVE_CHAR_X(ksi, "v12_p_pch", *uchar++);
1197         SAVE_CHAR_X(ksi, "v3_pch", *uchar++);
1198         SAVE_CHAR_X(ksi, "v5_pch", *uchar++);
1199         SAVE_CHAR_X(ksi, "p_fan", *uchar++);
1200 }
1201 #endif

1203 #ifdef __sparc
1204 static void
1205 save_fault_list(kstat_t *kp, ks_instance_t *ksi)
1206 {
1207         struct ft_list *fault;
1208         char    name[KSTAT_STRLEN + 7];
1209         int     i;

1211         for (i = 1, fault = (struct ft_list *)(kp->ks_data);
1212             i <= 999999 && i <= kp->ks_data_size / sizeof (struct ft_list);
1213             i++, fault++) {
1214                 (void) snprintf(name, sizeof (name), "unit_%d", i);
1215                 SAVE_INT32_X(ksi, name, fault->unit);
1216                 (void) snprintf(name, sizeof (name), "type_%d", i);
1217                 SAVE_INT32_X(ksi, name, fault->type);
1218                 (void) snprintf(name, sizeof (name), "fclass_%d", i);
1219                 SAVE_INT32_X(ksi, name, fault->fclass);
1220                 (void) snprintf(name, sizeof (name), "create_time_%d", i);
1221                 SAVE_HRTIME_X(ksi, name, fault->create_time);
1222                 (void) snprintf(name, sizeof (name), "msg_%d", i);
1223                 SAVE_STRING_X(ksi, name, faultp->msg);
1224         }
1225 }
1226 #endif

1228 static void
1229 save_named(kstat_t *kp, ks_instance_t *ksi)
1230 {
1231         kstat_named_t *knp;
1232         int     n;

1234         for (n = kp->ks_ndata, knp = KSTAT_NAMED_PTR(kp); n > 0; n--, knp++) {
1235                 switch (knp->data_type) {
1236                 case KSTAT_DATA_CHAR:
1237                         nvpair_insert(ksi, knp->name,
1238                             (ks_value_t *) &knp->value, KSTAT_DATA_CHAR);
1239                         break;
1240                 case KSTAT_DATA_INT32:
1241                         nvpair_insert(ksi, knp->name,
1242                             (ks_value_t *) &knp->value, KSTAT_DATA_INT32);
1243                         break;
1244                 case KSTAT_DATA_UINT32:
1245                         nvpair_insert(ksi, knp->name,
1246                             (ks_value_t *) &knp->value, KSTAT_DATA_UINT32);
1247                         break;
1248                 case KSTAT_DATA_INT64:
1249                         nvpair_insert(ksi, knp->name,
```



```
1250                             (ks_value_t *) &knp->value, KSTAT_DATA_INT64);
1251                         break;
1252                 case KSTAT_DATA_UINT64:
1253                         nvpair_insert(ksi, knp->name,
1254                             (ks_value_t *) &knp->value, KSTAT_DATA_UINT64);
1255                         break;
1256                 case KSTAT_DATA_STRING:
1257                         SAVE_STRING_X(ksi, knp->name, KSTAT_NAMED_STR_PTR(knp));
1258                         break;
1259                 default:
1260                         assert(B_FALSE); /* Invalid data type */
1261                         break;
1262                 }
1263         }
1264 }

1266 static void
1267 save_intr(kstat_t *kp, ks_instance_t *ksi)
1268 {
1269         kstat_intr_t *intr = KSTAT_INTR_PTR(kp);
1270         char    *intr_names[] = {"hard", "soft", "watchdog", "spurious",
1271             "multiple_service"};
1272         int     n;

1274         assert(sizeof (intr_names) / sizeof (char *) == KSTAT_NUM_INTRS);

1276         for (n = 0; n < KSTAT_NUM_INTRS; n++)
1277                 SAVE_UINT32_X(ksi, intr_names[n], intr->intrs[n]);
1278 }

1280 static void
1281 save_io(kstat_t *kp, ks_instance_t *ksi)
1282 {
1283         kstat_io_t      *ksio = KSTAT_IO_PTR(kp);

1285         SAVE_UINT64(ksi, ksio, nread);
1286         SAVE_UINT64(ksi, ksio, nwritten);
1287         SAVE_UINT32(ksi, ksio, reads);
1288         SAVE_UINT32(ksi, ksio, writes);
1289         SAVE_HRTIME(ksi, ksio, wtime);
1290         SAVE_HRTIME(ksi, ksio, wlentime);
1291         SAVE_HRTIME(ksi, ksio, wlastupdate);
1292         SAVE_HRTIME(ksi, ksio, rtime);
1293         SAVE_HRTIME(ksi, ksio, rlentime);
1294         SAVE_HRTIME(ksi, ksio, rlastupdate);
1295         SAVE_UINT32(ksi, ksio, wcnt);
1296         SAVE_UINT32(ksi, ksio, rcnt);
1297 }

1299 static void
1300 save_timer(kstat_t *kp, ks_instance_t *ksi)
1301 {
1302         kstat_timer_t   *ktimer = KSTAT_TIMER_PTR(kp);

1304         SAVE_STRING(ksi, ktimer, name);
1305         SAVE_UINT64(ksi, ktimer, num_events);
1306         SAVE_HRTIME(ksi, ktimer, elapsed_time);
1307         SAVE_HRTIME(ksi, ktimer, min_time);
1308         SAVE_HRTIME(ksi, ktimer, max_time);
1309         SAVE_HRTIME(ksi, ktimer, start_time);
1310         SAVE_HRTIME(ksi, ktimer, stop_time);
1311 }
1312 #endif /* ! codereview */
```

```
**********************************************************
    6655 Thu Aug 30 18:01:16 2012
new/usr/src/cmd/stat/kstat/kstat.h
749 "/usr/bin/kstat" should be rewritten in C
**********************************************************
```

```
/*
 * 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)                          \
{                                                       \
```

```
128         ks_value_t v;                                       \
129         v.ui32 = V;                                         \
130         nvpair_insert(I, N, &v, KSTAT_DATA_UINT32);         \
131 }

133 #define SAVE_UINT64_X(I, N, V)                              \
134 {                                                           \
135         ks_value_t v;                                       \
136         v.ui64 = V;                                         \
137         nvpair_insert(I, N, &v, KSTAT_DATA_UINT64);         \
138 }

140 #define SAVE_STRING_X(I, N, V)                              \
141 {                                                           \
142         ks_value_t v;                                       \
143         v.str.addr.ptr = safe_strdup(V);                    \
144         v.str.len = strlen(V);                              \
145         nvpair_insert(I, N, &v, KSTAT_DATA_STRING);         \
146 }

148 #define SAVE_CHAR_X(I, N, V)                                \
149 {                                                           \
150         ks_value_t v;                                       \
151         asprintf(&v.str.addr.ptr, "%c", V);                 \
152         v.str.len = 1;                                      \
153         nvpair_insert(I, N, &v, KSTAT_DATA_STRING);         \
154 }

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

160 #define JSON_FMT                                            \
161         "{\n\t\"module\": \"%s\",\n"                        \
162         "\t\"instance\": %d,\n"                             \
163         "\t\"name\": \"%s\",\n"                             \
164         "\t\"class\": \"%s\",\n"                            \
165         "\t\"statistics\": {\n"

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

171 typedef struct ks_instance {
172         list_node_t ks_next;
173         char    ks_name[KSTAT_STRLEN];
174         char    ks_module[KSTAT_STRLEN];
175         char    ks_class[KSTAT_STRLEN];
176         int     ks_instance;
177         list_t  ks_nvlist;
178 } ks_instance_t;

180 typedef struct ks_nvpair {
181         list_node_t nv_next;
182         char    name[KSTAT_STRLEN];
183         uchar_t data_type;
184         ks_value_t value;
185 } ks_nvpair_t;

187 typedef struct ks_selector {
188         list_node_t ks_next;
189         char    *ks_module;
190         char    *ks_instance;
191         char    *ks_name;
192         char    *ks_statistic;
193 } ks_selector_t;
```



```
195 static void     usage(void);
196 static int      compare_instances(ks_instance_t *, ks_instance_t *);
197 static void     nvpair_insert(ks_instance_t *, char *, ks_value_t *, uchar_t);
198 static ks_selector_t    *new_selector(void);
199 static void     ks_instances_read(kstat_ctl_t *);
200 static void     ks_value_print(ks_nvpair_t *);
201 static void     ks_instance_print(ks_instance_t *, ks_nvpair_t *);
202 static void     ks_instances_print(void);

204 /* Raw kstat readers */
205 static void     save_cpu_stat(kstat_t *, ks_instance_t *);
206 static void     save_var(kstat_t *, ks_instance_t *);
207 static void     save_ncstats(kstat_t *, ks_instance_t *);
208 static void     save_sysinfo(kstat_t *, ks_instance_t *);
209 static void     save_vminfo(kstat_t *, ks_instance_t *);
210 static void     save_nfs(kstat_t *, ks_instance_t *);
211 #ifdef __sparc
212 static void     save_sfmmu_global_stat(kstat_t *, ks_instance_t *);
213 static void     save_sfmmu_tsbsize_stat(kstat_t *, ks_instance_t *);
214 static void     save_simmstat(kstat_t *, ks_instance_t *);
215 /* Helper function for save_temperature() */
216 static char     *short_array_to_string(short *, int);
217 static void     save_temperature(kstat_t *, ks_instance_t *);
218 static void     save_temp_over(kstat_t *, ks_instance_t *);
219 static void     save_ps_shadow(kstat_t *, ks_instance_t *);
220 static void     save_fault_list(kstat_t *, ks_instance_t *);
221 #endif

223 /* Named kstat readers */
224 static void     save_named(kstat_t *, ks_instance_t *);
225 static void     save_intr(kstat_t *, ks_instance_t *);
226 static void     save_io(kstat_t *, ks_instance_t *);
227 static void     save_timer(kstat_t *, ks_instance_t *);

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

232 static struct {
233         kstat_raw_reader_t fn;
234         char *name;
235 } ks_raw_lookup[] = {
236         /* Function name                kstat name              */
237         {save_cpu_stat,                 "cpu_stat:cpu_stat"},
238         {save_var,                      "unix:var"},
239         {save_ncstats,                  "unix:ncstats"},
240         {save_sysinfo,                  "unix:sysinfo"},
241         {save_vminfo,                   "unix:vminfo"},
242         {save_nfs,                      "nfs:mntinfo"},
243 #ifdef __sparc
244         {save_sfmmu_global_stat,        "unix:sfmmu_global_stat"},
245         {save_sfmmu_tsbsize_stat,       "unix:sfmmu_tsbsize_stat"},
246         {save_simmstat,                 "unix:simm-status"},
247         {save_temperature,              "unix:temperature"},
248         {save_temp_over,                "unix:temperature override"},
249         {save_ps_shadow,                "unix:ps_shadow"},
250         {save_fault_list,               "unix:fault_list"},
251 #endif
252         {NULL, NULL},
253 };

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

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