

new/usr/src/cmd/stat/Makefile

1

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

```
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:
```

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

```
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 2009 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #
```

```
26 PROG = kstat
27 OBJS = kstat.o
28 SRCS =$(OBJS:%.o=%.c) $(COMMON_SRCS)
```

```
30 include $(SRC)/cmd/Makefile.cmd
31 include $(SRC)/cmd/stat/Makefile.stat
```

```
33 LDLIBS += -lavl -lcmdutils -ldevinfo -lgen -lkstat
34 CFLAGS += $(CCVERBOSE) -I${STATCOMMONDIR}
35 CERRWARN += _gcc=-Wno-uninitialized
36 CERRWARN += _gcc=-Wno-switch
37 CERRWARN += _gcc=-Wno-parentheses
38 FILEMODE= 0555
```

```
40 lint := LINTFLAGS = -muxs -I${STATCOMMONDIR}
```

```
42 .KEEP_STATE:
```

```
44 all: $(PROG)
```

```
46 install: all $(ROOTPROG)
```

```
48 $(PROG): $(OBJS) $(COMMON_OBJS)
49     $(LINK.c) -o $(PROG) $(OBJS) $(COMMON_OBJS) $(LDLIBS)
50     $(POST_PROCESS)
```

```
52 %.o : $(STATCOMMONDIR)/%.c
53     $(COMPILE.c) -o $@ $<
54     $(POST_PROCESS_O)
```

```
56 clean:
57     -$(RM) $(OBJS) $(COMMON_OBJS)
```

```
59 lint: lint_SRCS
```

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

new/usr/src/cmd/stat/kstat/kstat.c

1

```
*****
35416 Fri Nov 30 19:01:28 2012
new/usr/src/cmd/stat/kstat/kstat.c
749 "/usr/bin/kstat" should be rewritten in C
Reviewed by: Garrett D'Amore <garrett@damore.org>
Reviewed by: Brendan Gregg <brendan.gregg@joyent.com>
*****
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 /*
23  * Copyright (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright (c) 2012 David Hoepfner. All rights reserved.
25 */
26
27 /*
28  * Display kernel statistics
29  *
30  * This is a reimplementaion of the perl kstat command originally found
31  * under usr/src/cmd/kstat/kstat.pl
32  *
33  * Incompatibilities:
34  *   - perl regular expressions not longer supported
35  *   - options checking is stricter
36  *
37  * Flags added:
38  *   -C      similar to the -p option but value is separated by a colon
39  *   -h      display help
40  *   -j      json format
41  */
42
43 #include <assert.h>
44 #include <ctype.h>
45 #include <errno.h>
46 #include <kstat.h>
47 #include <langinfo.h>
48 #include <libgen.h>
49 #include <limits.h>
50 #include <locale.h>
51 #include <signal.h>
52 #include <stddef.h>
53 #include <stdio.h>
54 #include <stdlib.h>
55 #include <string.h>
56 #include <strings.h>
57 #include <time.h>
58 #include <unistd.h>
59 #include <sys/list.h>
```

new/usr/src/cmd/stat/kstat/kstat.c

2

```
60 #include <sys/time.h>
61 #include <sys/types.h>
62
63 #include "kstat.h"
64 #include "statcommon.h"
65
66 char *cmdname = "kstat"; /* Name of this command */
67 int caught_cont = 0; /* Have caught a SIGCONT */
68
69 static uint_t g_timestamp_fmt = NODATE;
70
71 /* Helper flag - header was printed already? */
72 static boolean_t g_headerflg;
73
74 /* Saved command line options */
75 static boolean_t g_cflg = B_FALSE;
76 static boolean_t g_jflg = B_FALSE;
77 static boolean_t g_lflg = B_FALSE;
78 static boolean_t g_pflg = B_FALSE;
79 static boolean_t g_qflg = B_FALSE;
80 static char *g_ks_class = "";
81
82 /* Return zero if a selector did match */
83 static int g_matched = 1;
84
85 /* Sorted list of kstat instances */
86 static list_t instances_list;
87 static list_t selector_list;
88
89 int
90 main(int argc, char **argv)
91 {
92     ks_selector_t *nselector;
93     ks_selector_t *uselector;
94     kstat_ctl_t *kc;
95     hrtime_t start_n;
96     hrtime_t period_n;
97     boolean_t errflg = B_FALSE;
98     boolean_t nsselflg = B_FALSE;
99     boolean_t usselflg = B_FALSE;
100    char *q;
101    int count = 1;
102    int infinite_cycles = 0;
103    int interval = 0;
104    int n = 0;
105    int c, m, tmp;
106
107    (void) setlocale(LC_ALL, "");
108    #if !defined(TEXT_DOMAIN) /* Should be defined by cc -D */
109    #define TEXT_DOMAIN "SYS_TEST" /* Use this only if it wasn't */
110    #endif
111    (void) textdomain(TEXT_DOMAIN);
112
113    /*
114     * Create the selector list and a dummy default selector to match
115     * everything. While we process the cmdline options we will add
116     * selectors to this list.
117     */
118    list_create(&selector_list, sizeof(ks_selector_t),
119              offsetof(ks_selector_t, ks_next));
120
121    nselector = new_selector();
122
123    /*
124     * Parse named command line arguments.
125     */
```

```

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

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

```

```

192         errflg = B_TRUE;
193     }

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

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

203     /*
204     * Consume the rest of the command line. Parsing the
205     * unnamed command line arguments.
206     */
207     while (argc-- > 0) {
208         errno = 0;
209         tmp = strtoul(*argv, &q, 10);
210         if (tmp == ULONG_MAX && errno == ERANGE) {
211             if (n == 0) {
212                 (void) fprintf(stderr, gettext(
213                     "Interval is too large\n"));
214             } else if (n == 1) {
215                 (void) fprintf(stderr, gettext(
216                     "Count is too large\n"));
217             }
218             usage();
219             exit(2);
220         }

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

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

257             if (m < 4) {

```

```

258         free(uselector);
259         usage();
260         exit(2);
261     }

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

293 /*
294  * Check if we founded a named selector on the cmdline.
295  */
296 if (useselfg) {
297     if (nselflg) {
298         (void) fprintf(stderr, gettext(
299             "module:instance:name:statistic and "
300             "-m -i -n -s are mutually exclusive"));
301         usage();
302         exit(2);
303     } else {
304         free(nselector);
305     }
306 } else {
307     list_insert_tail(&selector_list, nselector);
308 }

310 assert(!list_is_empty(&selector_list));

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

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

```

```

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

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

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

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

347     (void) kstat_close(kc);

349     return (g_matched);
350 }

352 /*
353  * Print usage.
354  */
355 static void
356 usage(void)
357 {
358     (void) fprintf(stderr, gettext(
359         "Usage:\n"
360         "kstat [ -Cjlpq ] [ -T d|u ] [ -c class ]\n"
361         "          [ -m module ] [ -i instance ] [ -n name ] [ -s statistic ]\n"
362         "          [ interval [ count ] ]\n"
363         "kstat [ -Cjlpq ] [ -T d|u ] [ -c class ]\n"
364         "          [ module:instance:name:statistic ... ]\n"
365         "          [ interval [ count ] ]\n"));
366 }

368 /*
369  * Sort compare function.
370  */
371 static int
372 compare_instances(ks_instance_t *l_arg, ks_instance_t *r_arg)
373 {
374     int     rval;

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

```

```

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

```

```

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

```

```

522         f++;
523         *t = *f;
524     }
525     *t++ = ':';

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

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

540     return (0);
541 }

543 /*
544  * Iterate over all kernel statistics and save matches.
545  */
546 static void
547 ks_instances_read(kstat_ctl_t *kc)
548 {
549     kstat_raw_reader_t save_raw = NULL;
550     kid_t id;
551     ks_selector_t *selector;
552     ks_instance_t *ksi;
553     ks_instance_t *tmp;
554     kstat_t *kp;
555     boolean_t skip;
556     char *ks_number;

558     for (kp = kc->kc_chain; kp != NULL; kp = kp->ks_next) {
559         /* Don't bother storing the kstat headers */
560         if (strncmp(kp->ks_name, "kstat_", 6) == 0) {
561             continue;
562         }

564         /* Don't bother storing raw stats we don't understand */
565         if (kp->ks_type == KSTAT_TYPE_RAW) {
566             save_raw = lookup_raw_kstat_fn(kp->ks_module,
567             kp->ks_name);
568             if (save_raw == NULL) {
569 #ifdef REPORT_UNKNOWN
570                 (void) fprintf(stderr,
571                 "Unknown kstat type %s:%d:%s - "
572                 "%d of size %d\n", kp->ks_module,
573                 kp->ks_instance, kp->ks_name,
574                 kp->ks_ndata, kp->ks_data_size);
575 #endif
576                 continue;
577             }
578         }

580         /*
581          * Iterate over the list of selectors and skip
582          * instances we dont want. We filter for statistics
583          * later, as we dont know them yet.
584          */
585         skip = B_FALSE;
586         (void) asprintf(&ks_number, "%d", kp->ks_instance);
587         selector = list_head(&selector_list);

```

```

588         while (selector != NULL) {
589             if (!(gmatch(kp->ks_module, selector->ks_module) != 0 &&
590             gmatch(ks_number, selector->ks_instance) != 0 &&
591             gmatch(kp->ks_name, selector->ks_name) != 0 &&
592             gmatch(kp->ks_class, g_ks_class))) {
593                 skip = B_TRUE;
594             }
595             selector = list_next(&selector_list, selector);
596         }

598         free(ks_number);

600         if (skip) {
601             continue;
602         }

604         /*
605          * Allocate a new instance and fill in the values
606          * we know so far.
607          */
608         ksi = (ks_instance_t *)malloc(sizeof (ks_instance_t));
609         if (ksi == NULL) {
610             perror("malloc");
611             exit(3);
612         }

614         list_link_init(&ksi->ks_next);

616         (void) strcpy(ksi->ks_module, kp->ks_module, KSTAT_STRLEN);
617         (void) strcpy(ksi->ks_name, kp->ks_name, KSTAT_STRLEN);
618         (void) strcpy(ksi->ks_class, kp->ks_class, KSTAT_STRLEN);

620         ksi->ks_instance = kp->ks_instance;
621         ksi->ks_snaptime = kp->ks_snaptime;
622         ksi->ks_type = kp->ks_type;

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

627         SAVE_HRTIME_X(ksi, "crttime", kp->ks_crttime);
628         SAVE_HRTIME_X(ksi, "snaptime", kp->ks_snaptime);
629         if (g_pflg) {
630             SAVE_STRING_X(ksi, "class", kp->ks_class);
631         }

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

638         list_insert_before(&instances_list, tmp, ksi);

640         /* Read the actual statistics */
641         id = kstat_read(kc, kp, NULL);
642         if (id == -1) {
643 #ifdef REPORT_UNKNOWN
644             perror("kstat_read");
645 #endif
646             continue;
647         }

649         switch (kp->ks_type) {
650         case KSTAT_TYPE_RAW:
651             save_raw(kp, ksi);
652             break;
653         case KSTAT_TYPE_NAMED:

```

```

654         save_named(kp, ksi);
655         break;
656     case KSTAT_TYPE_INTR:
657         save_intr(kp, ksi);
658         break;
659     case KSTAT_TYPE_IO:
660         save_io(kp, ksi);
661         break;
662     case KSTAT_TYPE_TIMER:
663         save_timer(kp, ksi);
664         break;
665     default:
666         assert(B_FALSE); /* Invalid type */
667         break;
668     }
669 }
670 }

672 /*
673  * Print the value of a name-value pair.
674  */
675 static void
676 ks_value_print(ks_nvpair_t *nvpair)
677 {
678     switch (nvpair->data_type) {
679     case KSTAT_DATA_CHAR:
680         (void) fprintf(stdout, "%s", nvpair->value.c);
681         break;
682     case KSTAT_DATA_INT32:
683         (void) fprintf(stdout, "%d", nvpair->value.i32);
684         break;
685     case KSTAT_DATA_UINT32:
686         (void) fprintf(stdout, "%u", nvpair->value.ui32);
687         break;
688     case KSTAT_DATA_INT64:
689         (void) fprintf(stdout, "%lld", nvpair->value.i64);
690         break;
691     case KSTAT_DATA_UINT64:
692         (void) fprintf(stdout, "%llu", nvpair->value.ui64);
693         break;
694     case KSTAT_DATA_STRING:
695         (void) fprintf(stdout, "%s", KSTAT_NAMED_STR_PTR(nvpair));
696         break;
697     case KSTAT_DATA_HRTIME:
698         if (nvpair->value.ui64 == 0)
699             (void) fprintf(stdout, "0");
700         else
701             (void) fprintf(stdout, "%.9f",
702                 nvpair->value.ui64 / 1000000000.0);
703         break;
704     default:
705         assert(B_FALSE);
706     }
707 }

709 /*
710  * Print a single instance.
711  */
712 static void
713 ks_instance_print(ks_instance_t *ksi, ks_nvpair_t *nvpair)
714 {
715     if (g_headerflg) {
716         if (!g_pflg) {
717             (void) fprintf(stdout, DFLT_FMT,
718                 ksi->ks_module, ksi->ks_instance,
719                 ksi->ks_name, ksi->ks_class);

```

```

720     }
721     g_headerflg = B_FALSE;
722 }

724     if (g_pflg) {
725         (void) fprintf(stdout, KS_PFMT,
726             ksi->ks_module, ksi->ks_instance,
727             ksi->ks_name, nvpair->name);
728         if (!g_lflg) {
729             (void) putchar(g_cflg ? ':': '\t');
730             ks_value_print(nvpair);
731         }
732     } else {
733         (void) fprintf(stdout, KS_DFMT, nvpair->name);
734         ks_value_print(nvpair);
735     }

737     (void) putchar('\n');
738 }

740 /*
741  * Print a single instance in JSON format.
742  */
743 static void
744 ks_instance_print_json(ks_instance_t *ksi, ks_nvpair_t *nvpair)
745 {
746     if (g_headerflg) {
747         (void) fprintf(stdout, JSON_FMT,
748             ksi->ks_module, ksi->ks_instance,
749             ksi->ks_name, ksi->ks_class,
750             ksi->ks_type);

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

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

760         g_headerflg = B_FALSE;
761     }

763     (void) fprintf(stdout, KS_JFMT, nvpair->name);
764     if (nvpair->data_type == KSTAT_DATA_STRING) {
765         (void) putchar('\n');
766         ks_value_print(nvpair);
767         (void) putchar('\n');
768     } else {
769         ks_value_print(nvpair);
770     }
771     if (nvpair != list_tail(&ksi->ks_nvlist))
772         (void) putchar(',');

774     (void) putchar('\n');
775 }

777 /*
778  * Print all instances.
779  */
780 static void
781 ks_instances_print(void)
782 {
783     ks_selector_t *selector;
784     ks_instance_t *ksi, *ktmp;
785     ks_nvpair_t *nvpair, *ntmp;

```



```

786 void (*ks_print_fn)(ks_instance_t *, ks_nvpair_t *);
788 if (g_timestamp_fmt != NODATE)
789     print_timestamp(g_timestamp_fmt);
791 if (g_jflg) {
792     ks_print_fn = &ks_instance_print_json;
793     (void) putchar('[');
794 } else {
795     ks_print_fn = &ks_instance_print;
796 }
798 /* Iterate over each selector */
799 selector = list_head(&selector_list);
800 while (selector != NULL) {
802     /* Iterate over each instance */
803     for (ksi = list_head(&instances_list); ksi != NULL;
804         ksi = list_next(&instances_list, ksi)) {
806         /* Finally iterate over each statistic */
807         g_headerflg = B_TRUE;
808         for (nvpair = list_head(&ksi->ks_nvlist);
809             nvpair != NULL;
810             nvpair = list_next(&ksi->ks_nvlist, nvpair)) {
811             if (gmatch(nvpair->name,
812                 selector->ks_statistic) == 0)
813                 continue;
815             g_matched = 0;
816             if (!g_qflg)
817                 (*ks_print_fn)(ksi, nvpair);
818         }
820         if (!g_headerflg) {
821             if (g_jflg) {
822                 (void) fprintf(stdout, "\t}\n");
823                 if (ksi != list_tail(&instances_list))
824                     (void) putchar(',');
825             } else if (!g_pflg) {
826                 (void) putchar('\n');
827             }
828         }
829     }
831     selector = list_next(&selector_list, selector);
832 }
834 if (g_jflg)
835     (void) fprintf(stdout, "]\n");
837 (void) fflush(stdout);
839 /* Free the instances list */
840 ksi = list_head(&instances_list);
841 while (ksi != NULL) {
842     nvpair = list_head(&ksi->ks_nvlist);
843     while (nvpair != NULL) {
844         ntmp = nvpair;
845         nvpair = list_next(&ksi->ks_nvlist, nvpair);
846         list_remove(&ksi->ks_nvlist, ntmp);
847         if (ntmp->data_type == KSTAT_DATA_STRING)
848             free(ntmp->value.str.addr.ptr);
849         free(ntmp);
850     }

```

```

852     ktmp = ksi;
853     ksi = list_next(&instances_list, ksi);
854     list_remove(&instances_list, ktmp);
855     list_destroy(&ktmp->ks_nvlist);
856     free(ktmp);
857 }
858 }
860 static void
861 save_cpu_stat(kstat_t *kp, ks_instance_t *ksi)
862 {
863     cpu_stat_t *stat;
864     cpu_sysinfo_t *sysinfo;
865     cpu_syswait_t *syswait;
866     cpu_vminfo_t *vminfo;
868     stat = (cpu_stat_t *) (kp->ks_data);
869     sysinfo = &stat->cpu_sysinfo;
870     syswait = &stat->cpu_syswait;
871     vminfo = &stat->cpu_vminfo;
873     SAVE_UINT32_X(ksi, "idle", sysinfo->cpu[CPU_IDLE]);
874     SAVE_UINT32_X(ksi, "user", sysinfo->cpu[CPU_USER]);
875     SAVE_UINT32_X(ksi, "kernel", sysinfo->cpu[CPU_KERNEL]);
876     SAVE_UINT32_X(ksi, "wait", sysinfo->cpu[CPU_WAIT]);
877     SAVE_UINT32_X(ksi, "wait_io", sysinfo->cpu[W_IO]);
878     SAVE_UINT32_X(ksi, "wait_swap", sysinfo->cpu[W_SWAP]);
879     SAVE_UINT32_X(ksi, "wait_pio", sysinfo->cpu[W_PIO]);
880     SAVE_UINT32(ksi, sysinfo, bread);
881     SAVE_UINT32(ksi, sysinfo, bwrite);
882     SAVE_UINT32(ksi, sysinfo, lread);
883     SAVE_UINT32(ksi, sysinfo, lwrite);
884     SAVE_UINT32(ksi, sysinfo, phread);
885     SAVE_UINT32(ksi, sysinfo, phwrite);
886     SAVE_UINT32(ksi, sysinfo, pswitch);
887     SAVE_UINT32(ksi, sysinfo, trap);
888     SAVE_UINT32(ksi, sysinfo, intr);
889     SAVE_UINT32(ksi, sysinfo, syscall);
890     SAVE_UINT32(ksi, sysinfo, sysread);
891     SAVE_UINT32(ksi, sysinfo, syswrite);
892     SAVE_UINT32(ksi, sysinfo, sysfork);
893     SAVE_UINT32(ksi, sysinfo, sysvfork);
894     SAVE_UINT32(ksi, sysinfo, sysexec);
895     SAVE_UINT32(ksi, sysinfo, readch);
896     SAVE_UINT32(ksi, sysinfo, writech);
897     SAVE_UINT32(ksi, sysinfo, rcvint);
898     SAVE_UINT32(ksi, sysinfo, xmtint);
899     SAVE_UINT32(ksi, sysinfo, mdmint);
900     SAVE_UINT32(ksi, sysinfo, rawch);
901     SAVE_UINT32(ksi, sysinfo, canch);
902     SAVE_UINT32(ksi, sysinfo, outch);
903     SAVE_UINT32(ksi, sysinfo, msg);
904     SAVE_UINT32(ksi, sysinfo, sema);
905     SAVE_UINT32(ksi, sysinfo, namei);
906     SAVE_UINT32(ksi, sysinfo, ufsiget);
907     SAVE_UINT32(ksi, sysinfo, ufsdirblk);
908     SAVE_UINT32(ksi, sysinfo, ufsipage);
909     SAVE_UINT32(ksi, sysinfo, ufsinopage);
910     SAVE_UINT32(ksi, sysinfo, inodeovf);
911     SAVE_UINT32(ksi, sysinfo, fileovf);
912     SAVE_UINT32(ksi, sysinfo, procovf);
913     SAVE_UINT32(ksi, sysinfo, intrthread);
914     SAVE_UINT32(ksi, sysinfo, intrblk);
915     SAVE_UINT32(ksi, sysinfo, idlethread);
916     SAVE_UINT32(ksi, sysinfo, inv_swch);
917     SAVE_UINT32(ksi, sysinfo, nthreads);

```

```

918     SAVE_UINT32(ksi, sysinfo, cpumigrate);
919     SAVE_UINT32(ksi, sysinfo, xcalls);
920     SAVE_UINT32(ksi, sysinfo, mutex_adenters);
921     SAVE_UINT32(ksi, sysinfo, rw_rdfails);
922     SAVE_UINT32(ksi, sysinfo, rw_wrfails);
923     SAVE_UINT32(ksi, sysinfo, modload);
924     SAVE_UINT32(ksi, sysinfo, modunload);
925     SAVE_UINT32(ksi, sysinfo, bawrite);
926 #ifdef STATISTICS /* see header file */
927     SAVE_UINT32(ksi, sysinfo, rw_enters);
928     SAVE_UINT32(ksi, sysinfo, win_uo_cnt);
929     SAVE_UINT32(ksi, sysinfo, win_uu_cnt);
930     SAVE_UINT32(ksi, sysinfo, win_so_cnt);
931     SAVE_UINT32(ksi, sysinfo, win_su_cnt);
932     SAVE_UINT32(ksi, sysinfo, win_suo_cnt);
933 #endif

935     SAVE_INT32(ksi, syswait, iowait);
936     SAVE_INT32(ksi, syswait, swap);
937     SAVE_INT32(ksi, syswait, physio);

939     SAVE_UINT32(ksi, vminfo, pgrec);
940     SAVE_UINT32(ksi, vminfo, pgfrec);
941     SAVE_UINT32(ksi, vminfo, pgin);
942     SAVE_UINT32(ksi, vminfo, pgpgin);
943     SAVE_UINT32(ksi, vminfo, pgout);
944     SAVE_UINT32(ksi, vminfo, pgpgout);
945     SAVE_UINT32(ksi, vminfo, swapin);
946     SAVE_UINT32(ksi, vminfo, pgswapin);
947     SAVE_UINT32(ksi, vminfo, swapout);
948     SAVE_UINT32(ksi, vminfo, pgswapout);
949     SAVE_UINT32(ksi, vminfo, zfod);
950     SAVE_UINT32(ksi, vminfo, dfree);
951     SAVE_UINT32(ksi, vminfo, scan);
952     SAVE_UINT32(ksi, vminfo, rev);
953     SAVE_UINT32(ksi, vminfo, hat_fault);
954     SAVE_UINT32(ksi, vminfo, as_fault);
955     SAVE_UINT32(ksi, vminfo, maj_fault);
956     SAVE_UINT32(ksi, vminfo, cow_fault);
957     SAVE_UINT32(ksi, vminfo, prot_fault);
958     SAVE_UINT32(ksi, vminfo, softlock);
959     SAVE_UINT32(ksi, vminfo, kernel_asflt);
960     SAVE_UINT32(ksi, vminfo, pgrrun);
961     SAVE_UINT32(ksi, vminfo, execpgin);
962     SAVE_UINT32(ksi, vminfo, execpgout);
963     SAVE_UINT32(ksi, vminfo, execfree);
964     SAVE_UINT32(ksi, vminfo, anonpgin);
965     SAVE_UINT32(ksi, vminfo, anonpgout);
966     SAVE_UINT32(ksi, vminfo, anonfree);
967     SAVE_UINT32(ksi, vminfo, fspgin);
968     SAVE_UINT32(ksi, vminfo, fspgout);
969     SAVE_UINT32(ksi, vminfo, fsfree);
970 }

972 static void
973 save_var(kstat_t *kp, ks_instance_t *ksi)
974 {
975     struct var      *var = (struct var *) (kp->ks_data);

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

979     SAVE_INT32(ksi, var, v_buf);
980     SAVE_INT32(ksi, var, v_call);
981     SAVE_INT32(ksi, var, v_proc);
982     SAVE_INT32(ksi, var, v_maxupttl);
983     SAVE_INT32(ksi, var, v_nglobpris);

```

```

984     SAVE_INT32(ksi, var, v_maxsyspri);
985     SAVE_INT32(ksi, var, v_clist);
986     SAVE_INT32(ksi, var, v_maxup);
987     SAVE_INT32(ksi, var, v_hbuf);
988     SAVE_INT32(ksi, var, v_hmask);
989     SAVE_INT32(ksi, var, v_pbuf);
990     SAVE_INT32(ksi, var, v_sptmap);
991     SAVE_INT32(ksi, var, v_maxpmem);
992     SAVE_INT32(ksi, var, v_autoup);
993     SAVE_INT32(ksi, var, v_bufhwm);
994 }

996 static void
997 save_ncstats(kstat_t *kp, ks_instance_t *ksi)
998 {
999     struct ncstats *ncstats = (struct ncstats *) (kp->ks_data);

1001     assert(kp->ks_data_size == sizeof (struct ncstats));

1003     SAVE_INT32(ksi, ncstats, hits);
1004     SAVE_INT32(ksi, ncstats, misses);
1005     SAVE_INT32(ksi, ncstats, enters);
1006     SAVE_INT32(ksi, ncstats, dbl_enters);
1007     SAVE_INT32(ksi, ncstats, long_enter);
1008     SAVE_INT32(ksi, ncstats, long_look);
1009     SAVE_INT32(ksi, ncstats, move_to_front);
1010     SAVE_INT32(ksi, ncstats, purges);
1011 }

1013 static void
1014 save_sysinfo(kstat_t *kp, ks_instance_t *ksi)
1015 {
1016     struct sysinfo_t *sysinfo = (struct sysinfo_t *) (kp->ks_data);

1018     assert(kp->ks_data_size == sizeof (struct sysinfo_t));

1020     SAVE_UINT32(ksi, sysinfo, updates);
1021     SAVE_UINT32(ksi, sysinfo, runque);
1022     SAVE_UINT32(ksi, sysinfo, runocc);
1023     SAVE_UINT32(ksi, sysinfo, swpque);
1024     SAVE_UINT32(ksi, sysinfo, swpocc);
1025     SAVE_UINT32(ksi, sysinfo, waiting);
1026 }

1028 static void
1029 save_vminfo(kstat_t *kp, ks_instance_t *ksi)
1030 {
1031     struct vminfo_t *vminfo = (struct vminfo_t *) (kp->ks_data);

1033     assert(kp->ks_data_size == sizeof (struct vminfo_t));

1035     SAVE_UINT64(ksi, vminfo, freemem);
1036     SAVE_UINT64(ksi, vminfo, swap_resv);
1037     SAVE_UINT64(ksi, vminfo, swap_alloc);
1038     SAVE_UINT64(ksi, vminfo, swap_avail);
1039     SAVE_UINT64(ksi, vminfo, swap_free);
1040     SAVE_UINT64(ksi, vminfo, updates);
1041 }

1043 static void
1044 save_nfs(kstat_t *kp, ks_instance_t *ksi)
1045 {
1046     struct mntinfo_kstat *mntinfo = (struct mntinfo_kstat *) (kp->ks_data);

1048     assert(kp->ks_data_size == sizeof (struct mntinfo_kstat));

```

```

1050     SAVE_STRING(ksi, mntinfo, mik_proto);
1051     SAVE_UINT32(ksi, mntinfo, mik_vers);
1052     SAVE_UINT32(ksi, mntinfo, mik_flags);
1053     SAVE_UINT32(ksi, mntinfo, mik_secm0d);
1054     SAVE_UINT32(ksi, mntinfo, mik_curread);
1055     SAVE_UINT32(ksi, mntinfo, mik_curwrite);
1056     SAVE_INT32(ksi, mntinfo, mik_timeo);
1057     SAVE_INT32(ksi, mntinfo, mik_retrans);
1058     SAVE_UINT32(ksi, mntinfo, mik_acregmin);
1059     SAVE_UINT32(ksi, mntinfo, mik_acregmax);
1060     SAVE_UINT32(ksi, mntinfo, mik_acdirmin);
1061     SAVE_UINT32(ksi, mntinfo, mik_acdirmax);
1062     SAVE_UINT32_X(ksi, "lookup_srtt", mntinfo->mik_timers[0].srtt);
1063     SAVE_UINT32_X(ksi, "lookup_deviate", mntinfo->mik_timers[0].deviate);
1064     SAVE_UINT32_X(ksi, "lookup_rtxcur", mntinfo->mik_timers[0].rtxcur);
1065     SAVE_UINT32_X(ksi, "read_srtt", mntinfo->mik_timers[1].srtt);
1066     SAVE_UINT32_X(ksi, "read_deviate", mntinfo->mik_timers[1].deviate);
1067     SAVE_UINT32_X(ksi, "read_rtxcur", mntinfo->mik_timers[1].rtxcur);
1068     SAVE_UINT32_X(ksi, "write_srtt", mntinfo->mik_timers[2].srtt);
1069     SAVE_UINT32_X(ksi, "write_deviate", mntinfo->mik_timers[2].deviate);
1070     SAVE_UINT32_X(ksi, "write_rtxcur", mntinfo->mik_timers[2].rtxcur);
1071     SAVE_UINT32(ksi, mntinfo, mik_noresponse);
1072     SAVE_UINT32(ksi, mntinfo, mik_failover);
1073     SAVE_UINT32(ksi, mntinfo, mik_remap);
1074     SAVE_STRING(ksi, mntinfo, mik_curserver);
1075 }

1077 #ifdef __sparc
1078 static void
1079 save_sfmmu_global_stat(kstat_t *kp, ks_instance_t *ksi)
1080 {
1081     struct sfmmu_global_stat *sfmmug =
1082         (struct sfmmu_global_stat *) (kp->ks_data);

1084     assert(kp->ks_data_size == sizeof (struct sfmmu_global_stat));

1086     SAVE_INT32(ksi, sfmmug, sf_tsb_exceptions);
1087     SAVE_INT32(ksi, sfmmug, sf_tsb_raise_exception);
1088     SAVE_INT32(ksi, sfmmug, sf_pagefaults);
1089     SAVE_INT32(ksi, sfmmug, sf_uhash_searches);
1090     SAVE_INT32(ksi, sfmmug, sf_uhash_links);
1091     SAVE_INT32(ksi, sfmmug, sf_khash_searches);
1092     SAVE_INT32(ksi, sfmmug, sf_khash_links);
1093     SAVE_INT32(ksi, sfmmug, sf_swapout);
1094     SAVE_INT32(ksi, sfmmug, sf_tsb_alloc);
1095     SAVE_INT32(ksi, sfmmug, sf_tsb_allocfail);
1096     SAVE_INT32(ksi, sfmmug, sf_tsb_sectsb_create);
1097     SAVE_INT32(ksi, sfmmug, sf_scd_1sttsb_alloc);
1098     SAVE_INT32(ksi, sfmmug, sf_scd_2ndtsb_alloc);
1099     SAVE_INT32(ksi, sfmmug, sf_scd_1sttsb_allocfail);
1100     SAVE_INT32(ksi, sfmmug, sf_scd_2ndtsb_allocfail);
1101     SAVE_INT32(ksi, sfmmug, sf_tteload8k);
1102     SAVE_INT32(ksi, sfmmug, sf_tteload64k);
1103     SAVE_INT32(ksi, sfmmug, sf_tteload512k);
1104     SAVE_INT32(ksi, sfmmug, sf_tteload4m);
1105     SAVE_INT32(ksi, sfmmug, sf_tteload32m);
1106     SAVE_INT32(ksi, sfmmug, sf_tteload256m);
1107     SAVE_INT32(ksi, sfmmug, sf_tsb_load8k);
1108     SAVE_INT32(ksi, sfmmug, sf_tsb_load4m);
1109     SAVE_INT32(ksi, sfmmug, sf_hblk_hit);
1110     SAVE_INT32(ksi, sfmmug, sf_hblk8_ncreate);
1111     SAVE_INT32(ksi, sfmmug, sf_hblk8_nalloc);
1112     SAVE_INT32(ksi, sfmmug, sf_hblk1_ncreate);
1113     SAVE_INT32(ksi, sfmmug, sf_hblk1_nalloc);
1114     SAVE_INT32(ksi, sfmmug, sf_hblk_slab_cnt);
1115     SAVE_INT32(ksi, sfmmug, sf_hblk_reserve_cnt);

```

```

1116     SAVE_INT32(ksi, sfmmug, sf_hblk_recurse_cnt);
1117     SAVE_INT32(ksi, sfmmug, sf_hblk_reserve_hit);
1118     SAVE_INT32(ksi, sfmmug, sf_get_free_success);
1119     SAVE_INT32(ksi, sfmmug, sf_get_free_throttle);
1120     SAVE_INT32(ksi, sfmmug, sf_get_free_fail);
1121     SAVE_INT32(ksi, sfmmug, sf_put_free_success);
1122     SAVE_INT32(ksi, sfmmug, sf_put_free_fail);
1123     SAVE_INT32(ksi, sfmmug, sf_pgcolor_conflict);
1124     SAVE_INT32(ksi, sfmmug, sf_uncache_conflict);
1125     SAVE_INT32(ksi, sfmmug, sf_unload_conflict);
1126     SAVE_INT32(ksi, sfmmug, sf_ism_uncache);
1127     SAVE_INT32(ksi, sfmmug, sf_ism_recache);
1128     SAVE_INT32(ksi, sfmmug, sf_recache);
1129     SAVE_INT32(ksi, sfmmug, sf_steal_count);
1130     SAVE_INT32(ksi, sfmmug, sf_pagesync);
1131     SAVE_INT32(ksi, sfmmug, sf_clrwrt);
1132     SAVE_INT32(ksi, sfmmug, sf_pagesync_invalid);
1133     SAVE_INT32(ksi, sfmmug, sf_kernel_xcalls);
1134     SAVE_INT32(ksi, sfmmug, sf_user_xcalls);
1135     SAVE_INT32(ksi, sfmmug, sf_tsb_grow);
1136     SAVE_INT32(ksi, sfmmug, sf_tsb_shrink);
1137     SAVE_INT32(ksi, sfmmug, sf_tsb_resize_failures);
1138     SAVE_INT32(ksi, sfmmug, sf_tsb_reloc);
1139     SAVE_INT32(ksi, sfmmug, sf_user_vtop);
1140     SAVE_INT32(ksi, sfmmug, sf_ctx_inv);
1141     SAVE_INT32(ksi, sfmmug, sf_tlb_reprog_pgsz);
1142     SAVE_INT32(ksi, sfmmug, sf_region_remap_demap);
1143     SAVE_INT32(ksi, sfmmug, sf_create_scd);
1144     SAVE_INT32(ksi, sfmmug, sf_join_scd);
1145     SAVE_INT32(ksi, sfmmug, sf_leave_scd);
1146     SAVE_INT32(ksi, sfmmug, sf_destroy_scd);
1147 }
1148 #endif

1150 #ifdef __sparc
1151 static void
1152 save_sfmmu_tsbsize_stat(kstat_t *kp, ks_instance_t *ksi)
1153 {
1154     struct sfmmu_tsbsize_stat *sfmmut;

1156     assert(kp->ks_data_size == sizeof (struct sfmmu_tsbsize_stat));
1157     sfmmut = (struct sfmmu_tsbsize_stat *) (kp->ks_data);

1159     SAVE_INT32(ksi, sfmmut, sf_tsbisz_8k);
1160     SAVE_INT32(ksi, sfmmut, sf_tsbisz_16k);
1161     SAVE_INT32(ksi, sfmmut, sf_tsbisz_32k);
1162     SAVE_INT32(ksi, sfmmut, sf_tsbisz_64k);
1163     SAVE_INT32(ksi, sfmmut, sf_tsbisz_128k);
1164     SAVE_INT32(ksi, sfmmut, sf_tsbisz_256k);
1165     SAVE_INT32(ksi, sfmmut, sf_tsbisz_512k);
1166     SAVE_INT32(ksi, sfmmut, sf_tsbisz_1m);
1167     SAVE_INT32(ksi, sfmmut, sf_tsbisz_2m);
1168     SAVE_INT32(ksi, sfmmut, sf_tsbisz_4m);
1169 }
1170 #endif

1172 #ifdef __sparc
1173 static void
1174 save_simmstat(kstat_t *kp, ks_instance_t *ksi)
1175 {
1176     uchar_t *simmstat;
1177     char *simm_buf;
1178     char *list = NULL;
1179     int i;

1181     assert(kp->ks_data_size == sizeof (uchar_t) * SIMM_COUNT);

```

```

1183     for (i = 0, simmstat = (uchar_t *) (kp->ks_data); i < SIMM_COUNT - 1;
1184          i++, simmstat++) {
1185         if (list == NULL) {
1186             (void) asprintf(&simm_buf, "%d,", *simmstat);
1187         } else {
1188             (void) asprintf(&simm_buf, "%s%d,", list, *simmstat);
1189             free(list);
1190         }
1191         list = simm_buf;
1192     }
1194     (void) asprintf(&simm_buf, "%s%d", list, *simmstat);
1195     SAVE_STRING_X(ksi, "status", simm_buf);
1196     free(list);
1197     free(simm_buf);
1198 }
1199 #endif

1201 #ifdef __sparc
1202 /*
1203  * Helper function for save_temperature().
1204  */
1205 static char *
1206 short_array_to_string(short *shortp, int len)
1207 {
1208     char *list = NULL;
1209     char *list_buf;
1211     for (; len > 1; len--, shortp++) {
1212         if (list == NULL) {
1213             (void) asprintf(&list_buf, "%d,", *shortp);
1214         } else {
1215             (void) asprintf(&list_buf, "%s%d,", list, *shortp);
1216             free(list);
1217         }
1218         list = list_buf;
1219     }
1221     (void) asprintf(&list_buf, "%s%s", list, *shortp);
1222     free(list);
1223     return (list_buf);
1224 }

1226 static void
1227 save_temperature(kstat_t *kp, ks_instance_t *ksi)
1228 {
1229     struct temp_stats *temps = (struct temp_stats *) (kp->ks_data);
1230     char *buf;
1231     int n = 1;
1233     assert(kp->ks_data_size == sizeof (struct temp_stats));
1235     SAVE_UINT32(ksi, temps, index);
1237     buf = short_array_to_string(temps->l1, L1_SZ);
1238     SAVE_STRING_X(ksi, "l1", buf);
1239     free(buf);
1241     buf = short_array_to_string(temps->l2, L2_SZ);
1242     SAVE_STRING_X(ksi, "l2", buf);
1243     free(buf);
1245     buf = short_array_to_string(temps->l3, L3_SZ);
1246     SAVE_STRING_X(ksi, "l3", buf);
1247     free(buf);

```

```

1249     buf = short_array_to_string(temps->l4, L4_SZ);
1250     SAVE_STRING_X(ksi, "l4", buf);
1251     free(buf);
1253     buf = short_array_to_string(temps->l5, L5_SZ);
1254     SAVE_STRING_X(ksi, "l5", buf);
1255     free(buf);
1257     SAVE_INT32(ksi, temps, max);
1258     SAVE_INT32(ksi, temps, min);
1259     SAVE_INT32(ksi, temps, state);
1260     SAVE_INT32(ksi, temps, temp_cnt);
1261     SAVE_INT32(ksi, temps, shutdown_cnt);
1262     SAVE_INT32(ksi, temps, version);
1263     SAVE_INT32(ksi, temps, trend);
1264     SAVE_INT32(ksi, temps, override);
1265 }
1266 #endif

1268 #ifdef __sparc
1269 static void
1270 save_temp_over(kstat_t *kp, ks_instance_t *ksi)
1271 {
1272     short *sh = (short *) (kp->ks_data);
1273     char *value;
1275     assert(kp->ks_data_size == sizeof (short));
1277     (void) asprintf(&value, "%hu", *sh);
1278     SAVE_STRING_X(ksi, "override", value);
1279     free(value);
1280 }
1281 #endif

1283 #ifdef __sparc
1284 static void
1285 save_ps_shadow(kstat_t *kp, ks_instance_t *ksi)
1286 {
1287     uchar_t *uchar = (uchar_t *) (kp->ks_data);
1289     assert(kp->ks_data_size == SYS_PS_COUNT);
1291     SAVE_CHAR_X(ksi, "core_0", *uchar++);
1292     SAVE_CHAR_X(ksi, "core_1", *uchar++);
1293     SAVE_CHAR_X(ksi, "core_2", *uchar++);
1294     SAVE_CHAR_X(ksi, "core_3", *uchar++);
1295     SAVE_CHAR_X(ksi, "core_4", *uchar++);
1296     SAVE_CHAR_X(ksi, "core_5", *uchar++);
1297     SAVE_CHAR_X(ksi, "core_6", *uchar++);
1298     SAVE_CHAR_X(ksi, "core_7", *uchar++);
1299     SAVE_CHAR_X(ksi, "pps_0", *uchar++);
1300     SAVE_CHAR_X(ksi, "clk_33", *uchar++);
1301     SAVE_CHAR_X(ksi, "clk_50", *uchar++);
1302     SAVE_CHAR_X(ksi, "v5_p", *uchar++);
1303     SAVE_CHAR_X(ksi, "v12_p", *uchar++);
1304     SAVE_CHAR_X(ksi, "v5_aux", *uchar++);
1305     SAVE_CHAR_X(ksi, "v5_p_pch", *uchar++);
1306     SAVE_CHAR_X(ksi, "v12_p_pch", *uchar++);
1307     SAVE_CHAR_X(ksi, "v3_pch", *uchar++);
1308     SAVE_CHAR_X(ksi, "v5_pch", *uchar++);
1309     SAVE_CHAR_X(ksi, "p_fan", *uchar++);
1310 }
1311 #endif

1313 #ifdef __sparc

```

```

1314 static void
1315 save_fault_list(kstat_t *kp, ks_instance_t *ksi)
1316 {
1317     struct ft_list *fault;
1318     char name[KSTAT_STRLEN + 7];
1319     int i;

1321     for (i = 1, fault = (struct ft_list *) (kp->ks_data);
1322          i <= 999999 && i <= kp->ks_data_size / sizeof (struct ft_list);
1323          i++, fault++) {
1324         (void) snprintf(name, sizeof (name), "unit_%d", i);
1325         SAVE_UINT32_X(ksi, name, fault->unit);
1326         (void) snprintf(name, sizeof (name), "type_%d", i);
1327         SAVE_UINT32_X(ksi, name, fault->type);
1328         (void) snprintf(name, sizeof (name), "fclass_%d", i);
1329         SAVE_UINT32_X(ksi, name, fault->fclass);
1330         (void) snprintf(name, sizeof (name), "create_time_%d", i);
1331         SAVE_HRTIME_X(ksi, name, fault->create_time);
1332         (void) snprintf(name, sizeof (name), "msg_%d", i);
1333         SAVE_STRING_X(ksi, name, faultp->msg);
1334     }
1335 }
1336 #endif

1338 static void
1339 save_named(kstat_t *kp, ks_instance_t *ksi)
1340 {
1341     kstat_named_t *knp;
1342     int n;

1344     for (n = kp->ks_ndata, knp = KSTAT_NAMED_PTR(kp); n > 0; n--, knp++) {
1345         switch (knp->data_type) {
1346             case KSTAT_DATA_CHAR:
1347                 nvpair_insert(ksi, knp->name,
1348                     (ks_value_t *)&knp->value, KSTAT_DATA_CHAR);
1349                 break;
1350             case KSTAT_DATA_INT32:
1351                 nvpair_insert(ksi, knp->name,
1352                     (ks_value_t *)&knp->value, KSTAT_DATA_INT32);
1353                 break;
1354             case KSTAT_DATA_UINT32:
1355                 nvpair_insert(ksi, knp->name,
1356                     (ks_value_t *)&knp->value, KSTAT_DATA_UINT32);
1357                 break;
1358             case KSTAT_DATA_INT64:
1359                 nvpair_insert(ksi, knp->name,
1360                     (ks_value_t *)&knp->value, KSTAT_DATA_INT64);
1361                 break;
1362             case KSTAT_DATA_UINT64:
1363                 nvpair_insert(ksi, knp->name,
1364                     (ks_value_t *)&knp->value, KSTAT_DATA_UINT64);
1365                 break;
1366             case KSTAT_DATA_STRING:
1367                 SAVE_STRING_X(ksi, knp->name, KSTAT_NAMED_STR_PTR(knp));
1368                 break;
1369             default:
1370                 assert(B_FALSE); /* Invalid data type */
1371                 break;
1372         }
1373     }
1374 }

1376 static void
1377 save_intr(kstat_t *kp, ks_instance_t *ksi)
1378 {
1379     kstat_intr_t *intr = KSTAT_INTR_PTR(kp);

```

```

1380     char *intr_names[] = {"hard", "soft", "watchdog", "spurious",
1381         "multiple_service"};
1382     int n;

1384     for (n = 0; n < KSTAT_NUM_INTRS; n++)
1385         SAVE_UINT32_X(ksi, intr_names[n], intr->intrs[n]);
1386 }

1388 static void
1389 save_io(kstat_t *kp, ks_instance_t *ksi)
1390 {
1391     kstat_io_t *ksio = KSTAT_IO_PTR(kp);

1393     SAVE_UINT64(ksi, ksio, nread);
1394     SAVE_UINT64(ksi, ksio, nwritten);
1395     SAVE_UINT32(ksi, ksio, reads);
1396     SAVE_UINT32(ksi, ksio, writes);
1397     SAVE_HRTIME(ksi, ksio, wtime);
1398     SAVE_HRTIME(ksi, ksio, wlentime);
1399     SAVE_HRTIME(ksi, ksio, wlastupdate);
1400     SAVE_HRTIME(ksi, ksio, rtime);
1401     SAVE_HRTIME(ksi, ksio, rlentime);
1402     SAVE_HRTIME(ksi, ksio, rlastupdate);
1403     SAVE_UINT32(ksi, ksio, wcnt);
1404     SAVE_UINT32(ksi, ksio, rcnt);
1405 }

1407 static void
1408 save_timer(kstat_t *kp, ks_instance_t *ksi)
1409 {
1410     kstat_timer_t *ktimer = KSTAT_TIMER_PTR(kp);

1412     SAVE_STRING(ksi, ktimer, name);
1413     SAVE_UINT64(ksi, ktimer, num_events);
1414     SAVE_HRTIME(ksi, ktimer, elapsed_time);
1415     SAVE_HRTIME(ksi, ktimer, min_time);
1416     SAVE_HRTIME(ksi, ktimer, max_time);
1417     SAVE_HRTIME(ksi, ktimer, start_time);
1418     SAVE_HRTIME(ksi, ktimer, stop_time);
1419 }
1420 #endif /* ! codereview */

```

```

*****
6801 Fri Nov 30 19:01:28 2012
new/usr/src/cmd/stat/kstat/kstat.h
749 "/usr/bin/kstat" should be rewritten in C
Reviewed by: Garrett D'Amore <garrett@damore.org>
Reviewed by: Brendan Gregg <brendan.gregg@joyent.com>
*****
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 * Copyright 2012 David Hoepfner. All rights reserved.
24 */

26 #ifndef _STAT_KSTAT_H
27 #define _STAT_KSTAT_H

29 /*
30  * Structures needed by the kstat reader functions.
31  */
32 #include <sys/var.h>
33 #include <sys/utsname.h>
34 #include <sys/sysinfo.h>
35 #include <sys/flock.h>
36 #include <sys/dnnc.h>
37 #include <nfs/nfs.h>
38 #include <nfs/nfs_clnt.h>

40 #ifdef __sparc
41 #include <vm/hat_sfmmu.h>
42 #include <sys/simmstat.h>
43 #include <sys/sysctrl.h>
44 #include <sys/fhc.h>
45 #endif

47 #define KSTAT_DATA_HRTIME      (KSTAT_DATA_STRING + 1)

49 typedef union ks_value {
50     char      c[16];
51     int32_t   i32;
52     uint32_t  ui32;
53     struct {
54         union {
55             char      *ptr;
56             char      __pad[8];
57         } addr;
58         uint32_t      len;
59     } str;

```

```

61     int64_t      i64;
62     uint64_t     ui64;
63 } ks_value_t;

65 #define SAVE_HRTIME(I, S, N)      \
66 {                                \
67     ks_value_t v;                \
68     v.ui64 = S->N;                \
69     nvpair_insert(I, #N, &v, KSTAT_DATA_UINT64); \
70 }

72 #define SAVE_INT32(I, S, N)      \
73 {                                \
74     ks_value_t v;                \
75     v.i32 = S->N;                \
76     nvpair_insert(I, #N, &v, KSTAT_DATA_INT32); \
77 }

79 #define SAVE_UINT32(I, S, N)     \
80 {                                \
81     ks_value_t v;                \
82     v.ui32 = S->N;                \
83     nvpair_insert(I, #N, &v, KSTAT_DATA_UINT32); \
84 }

86 #define SAVE_INT64(I, S, N)      \
87 {                                \
88     ks_value_t v;                \
89     v.i64 = S->N;                \
90     nvpair_insert(I, #N, &v, KSTAT_DATA_INT64); \
91 }

93 #define SAVE_UINT64(I, S, N)     \
94 {                                \
95     ks_value_t v;                \
96     v.ui64 = S->N;                \
97     nvpair_insert(I, #N, &v, KSTAT_DATA_UINT64); \
98 }

100 /*
101  * We dont want const "strings" because we free
102  * the instances later.
103  */
104 #define SAVE_STRING(I, S, N)      \
105 {                                \
106     ks_value_t v;                \
107     v.str.addr.ptr = safe_strdup(S->N); \
108     v.str.len = strlen(S->N);        \
109     nvpair_insert(I, #N, &v, KSTAT_DATA_STRING); \
110 }

112 #define SAVE_HRTIME_X(I, N, V)    \
113 {                                \
114     ks_value_t v;                \
115     v.ui64 = V;                  \
116     nvpair_insert(I, N, &v, KSTAT_DATA_HRTIME); \
117 }

119 #define SAVE_INT32_X(I, N, V)     \
120 {                                \
121     ks_value_t v;                \
122     v.i32 = V;                  \
123     nvpair_insert(I, N, &v, KSTAT_DATA_INT32); \
124 }

```

```

126 #define SAVE_UINT32_X(I, N, V) \
127 { \
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\"type\": %d, \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     uchar_t ks_type;
178     hrtime_t ks_snaptime;
179     list_t ks_nvlist;
180 } ks_instance_t;

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

189 typedef struct ks_selector {
190     list_node_t ks_next;
191     char *ks_module;

```

```

192     char *ks_instance;
193     char *ks_name;
194     char *ks_statistic;
195 } ks_selector_t;

197 static void usage(void);
198 static int compare_instances(ks_instance_t *, ks_instance_t *);
199 static void nvpair_insert(ks_instance_t *, char *, ks_value_t *, uchar_t);
200 static ks_selector_t *new_selector(void);
201 static void ks_instances_read(kstat_ctl_t *);
202 static void ks_value_print(ks_nvpair_t *);
203 static void ks_instance_print(ks_instance_t *, ks_nvpair_t *);
204 static void ks_instances_print(void);
205 static char *ks_safe_strdup(char *);
206 static void ks_sleep_until(hrtime_t *, hrtime_t, int, int *);

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

227 /* Named kstat readers */
228 static void save_named(kstat_t *, ks_instance_t *);
229 static void save_intr(kstat_t *, ks_instance_t *);
230 static void save_io(kstat_t *, ks_instance_t *);
231 static void save_timer(kstat_t *, ks_instance_t *);

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

236 static struct {
237     kstat_raw_reader_t fn;
238     char *name;
239 } ks_raw_lookup[] = {
240     /* Function name kstat name */
241     {save_cpu_stat, "cpu_stat:cpu_stat"},
242     {save_var, "unix:var"},
243     {save_ncstats, "unix:ncstats"},
244     {save_sysinfo, "unix:sysinfo"},
245     {save_vminfo, "unix:vminfo"},
246     {save_nfs, "nfs:mntinfo"},
247 #ifdef __sparc
248     {save_sfmmu_global_stat, "unix:sfmmu_global_stat"},
249     {save_sfmmu_tsbsize_stat, "unix:sfmmu_tsbsize_stat"},
250     {save_simmstat, "unix:simm-status"},
251     {save_temperature, "unix:temperature"},
252     {save_temp_over, "unix:temperature override"},
253     {save_ps_shadow, "unix:ps_shadow"},
254     {save_fault_list, "unix:fault_list"},
255 #endif
256     {NULL, NULL},
257 };

```

```
259 static kstat_raw_reader_t      lookup_raw_kstat_fn(char *, char *);
261 #endif /* _STAT_KSTAT_H */
262 #endif /* ! codereview */
```



```

*****
8929 Fri Nov 30 19:01:28 2012
new/usr/src/man/man1m/kstat.1m
749 "/usr/bin/kstat" should be rewritten in C
Reviewed by: Garrett D'Amore <garrett@damore.org>
Reviewed by: Brendan Gregg <brendan.gregg@joyent.com>
*****
1 \" te
2 .\" Copyright (c) 2000, Sun Microsystems, Inc. All Rights Reserved
3 .\" The contents of this file are subject to the terms of the Common Development
4 .\" See the License for the specific language governing permissions and limitat
5 .\" the fields enclosed by brackets \"[]\" replaced with your own identifying info
6 .TH KSTAT 1M \"Nov 22, 2012\"
6 .TH KSTAT 1M \"Mar 23, 2009\"
7 .SH NAME
8 kstat \- display kernel statistics
9 .SH SYNOPSIS
10 .LP
11 .nf
12 \fBkstat\fR [\fB-Cjlpq\fR] [\fB-T\fR u | d ] [\fB-c\fR \fIclass\fR] [\fB-m\fR \fI
12 \fBkstat\fR [\fB-lpq\fR] [\fB-T\fR u | d ] [\fB-c\fR \fIclass\fR] [\fB-m\fR \fIm
13 [\fB-i\fR \fIinstance\fR] [\fB-n\fR \fIname\fR] [\fB-s\fR \fIstatistic\fR]
14 [interval [count]]
15 .fi
17 .LP
18 .nf
19 \fBkstat\fR [\fB-Cjlpq\fR] [\fB-T\fR u | d ] [\fB-c\fR \fIclass\fR]
19 \fBkstat\fR [\fB-lpq\fR] [\fB-T\fR u | d ] [\fB-c\fR \fIclass\fR]
20 [\fImodule\fR:\fIinstance\fR:\fIname\fR:\fIstatistic\fR]...
21 [interval [count]]
22 .fi
24 .SH DESCRIPTION
25 .sp
26 .LP
27 The \fBkstat\fR utility examines the available kernel statistics, or kstats, on
28 the system and reports those statistics which match the criteria specified on
29 the command line. Each matching statistic is printed with its module, instance,
30 and name fields, as well as its actual value.
31 .sp
32 .LP
33 Kernel statistics may be published by various kernel subsystems, such as
34 drivers or loadable modules; each kstat has a module field that denotes its
35 publisher. Since each module might have countable entities (such as multiple
36 disks associated with the \fBsd\fR(7D) driver) for which it wishes to report
37 statistics, the kstat also has an instance field to index the statistics for
38 each entity; kstat instances are numbered starting from zero. Finally, the
39 kstat is given a name unique within its module.
40 .sp
41 .LP
42 Each kstat may be a special kstat type, an array of name-value pairs, or raw
43 data. In the name-value case, each reported value is given a label, which we
44 refer to as the statistic. Known raw and special kstats are given statistic
45 labels for each of their values by \fBkstat\fR; thus, all published values can
46 be referenced as \fImodule\fR:\fIinstance\fR:\fIname\fR:\fIstatistic\fR.
47 .sp
48 .LP
49 When invoked without any module operands or options, kstat will match all
50 defined statistics on the system. Example invocations are provided below. All
51 times are displayed as fractional seconds since system boot.
52 .SH OPTIONS
53 .sp
54 .LP
55 The tests specified by the following options are logically ANDed, and all
56 matching kstats will be selected. A regular expression containing shell

```

```

57 metacharacters must be protected from the shell by enclosing it with the
58 appropriate quotes.
59 .sp
60 .LP
61 The argument for the \fB-c\fR, \fB-i\fR, \fB-m\fR, \fB-n\fR, and \fB-s\fR
62 options may be specified as a shell glob pattern.
63 .sp
64 .ne 2
65 .na
66 \fB\FB-C\FR\FR
67 .ad
68 .RS 16n
69 Displays output in parseable format with a colon as separator.
70 .RE
62 options may be specified as a shell glob pattern, or a Perl regular expression
63 enclosed in '/' characters.
72 .sp
73 .ne 2
74 .na
75 \fB\FB-c\FR \fIclass\FR\FR
76 .ad
77 .RS 16n
78 Displays only kstats that match the specified class. \fIclass\FR is a
79 kernel-defined string which classifies the "type" of the kstat.
80 .RE
82 .sp
83 .ne 2
84 .na
85 \fB\FB-i\FR \fIinstance\FR\FR
86 .ad
87 .RS 16n
88 Displays only kstats that match the specified instance.
89 .RE
91 .sp
92 .ne 2
93 .na
94 \fB\FB-j\FR\FR
95 .ad
96 .RS 16n
97 Displays output in JSON format.
98 .RE
100 .sp
101 .ne 2
102 .na
103 #endif /* ! codereview */
104 \fB\FB-l\FR\FR
105 .ad
106 .RS 16n
107 Lists matching kstat names without displaying values.
108 .RE
110 .sp
111 .ne 2
112 .na
113 \fB\FB-m\FR \fImodule\FR\FR
114 .ad
115 .RS 16n
116 Displays only kstats that match the specified module.
117 .RE
119 .sp
120 .ne 2

```

```

121 .na
122 \fB\fB-n\fR \fIname\fR\fR
123 .ad
124 .RS 16n
125 Displays only kstats that match the specified name.
126 .RE

128 .sp
129 .ne 2
130 .na
131 \fB\fB-p\fR\fR
132 .ad
133 .RS 16n
134 Displays output in parseable format. All example output in this document is
135 given in this format. If this option is not specified, \fBkstat\fR produces
136 output in a human-readable, table format.
137 .RE

139 .sp
140 .ne 2
141 .na
142 \fB\fB-q\fR\fR
143 .ad
144 .RS 16n
145 Displays no output, but return appropriate exit status for matches against
146 given criteria.
147 .RE

149 .sp
150 .ne 2
151 .na
152 \fB\fB-s\fR \fIstatistic\fR\fR
153 .ad
154 .RS 16n
155 Displays only kstats that match the specified statistic.
156 .RE

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

169 .SH OPERANDS
170 .sp
171 .LP
172 The following operands are supported:
173 .sp
174 .ne 2
175 .na
176 \fB\fImodule\fR:\fIinstance\fR:\fIname\fR:\fIstatistic\fR\fR
177 .ad
178 .sp .6
179 .RS 4n
180 Alternate method of specifying module, instance, name, and statistic as
181 described above. Each of the module, instance, name, or statistic specifiers
182 may be a shell glob pattern.
183 It is possible to use both specifier types within a single operand.
184 86 may be a shell glob pattern or a Perl regular expression enclosed by ''
185 87 characters. It is possible to use both specifier types within a single operand.
184 Leaving a specifier empty is equivalent to using the '*' glob pattern for that

```

```

185 specifier.
186 .RE

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

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

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

224 unix:0:system_misc:avenrun_15min      3
225 unix:0:system_misc:avenrun_1min      4
226 unix:0:system_misc:avenrun_5min      2
227 .fi
228 .in -2
229 .sp

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

239 cpu_stat:0:cpu_stat0:intr      29682330
240 cpu_stat:0:cpu_stat0:intrblk    87
241 cpu_stat:0:cpu_stat0:intrthread 15054222
242 cpu_stat:1:cpu_stat1:intr      426073
243 cpu_stat:1:cpu_stat1:intrblk    51
244 cpu_stat:1:cpu_stat1:intrthread 289668
245 cpu_stat:2:cpu_stat2:intr      134160
246 cpu_stat:2:cpu_stat2:intrblk    0
247 cpu_stat:2:cpu_stat2:intrthread 131
248 cpu_stat:3:cpu_stat3:intr      196566

```

```

249 cpu_stat:3:cpu_stat3:intrblk      30
250 cpu_stat:3:cpu_stat3:intrthread  59626
251 .fi
252 .in -2
253 .sp

255 .LP
256 \fBExample 3 \fRUsing the \fBkstat\fR Command
257 .sp
258 .in +2
259 .nf
260 example$ \fBkstat -p :::state '::::avenrun*'\fR
166 example$ \fBkstat -p :::state :::^avenrun/\fR

262 cpu_info:0:cpu_info0:state        on-line
263 cpu_info:1:cpu_info1:state        on-line
264 cpu_info:2:cpu_info2:state        on-line
265 cpu_info:3:cpu_info3:state        on-line
266 unix:0:system_misc:avenrun_15min  4
267 unix:0:system_misc:avenrun_1min  10
268 unix:0:system_misc:avenrun_5min  3
269 .fi
270 .in -2
271 .sp

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

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
290 .fi
291 .in -2
292 .sp

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

305 Thu Jul 22 19:39:55 1999
306 unix:0:system_misc:avenrun_15min  12
307 unix:0:system_misc:avenrun_1min  0
308 unix:0:system_misc:avenrun_5min  11
309 .fi
310 .in -2
311 .sp

313 .LP

```

```

314 \fBExample 6 \fRUsing the \fBkstat\fR Command
315 .sp
316 .in +2
317 .nf
318 example$ \fBkstat -p -T u 'unix:0:system_misc:avenrun*'\fR
319 932668656
320 unix:0:system_misc:avenrun_15min  14
321 unix:0:system_misc:avenrun_1min  5
322 unix:0:system_misc:avenrun_5min  18
323 .fi
324 .in -2
325 .sp

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

340 .sp
341 .ne 2
342 .na
343 \fB\fB1\fR\fR
344 .ad
345 .RS 5n
346 No statistics were matched.
347 .RE

349 .sp
350 .ne 2
351 .na
352 \fB\fB2\fR\fR
353 .ad
354 .RS 5n
355 Invalid command line options were specified.
356 .RE

358 .sp
359 .ne 2
360 .na
361 \fB\fB3\fR\fR
362 .ad
363 .RS 5n
364 A fatal error occurred.
365 .RE

367 .SH FILES
368 .sp
369 .ne 2
370 .na
371 \fB\fB/dev/kstat\fR\fR
372 .ad
373 .RS 14n
374 kernel statistics driver
375 .RE

377 .SH SEE ALSO
378 .sp
379 .LP

```

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