

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

1

```
*****
36592 Wed Mar 13 17:08:29 2013
new/usr/src/cmd/stat/kstat/kstat.c
3623 kstat must accept partial stat specification
*****
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) 2013 David Hoepfner. All rights reserved.
25  * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
26  */
27
28 /*
29  * Display kernel statistics
30  *
31  * This is a reimplementaion of the perl kstat command originally found
32  * under usr/src/cmd/kstat/kstat.pl
33  *
34  * Incompatibilities:
35  *   - perl regular expressions replaced with extended REs bracketed by '/'
36  *   - options checking is stricter
37  *
38  * Flags added:
39  *   -C similar to the -p option but value is separated by a colon
40  *   -h display help
41  *   -j json format
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>
60 #include <sys/time.h>
```

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

2

```
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_header_flg;
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 ks_pattern_t g_ks_class = {"*", 0};
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 nselflg = B_FALSE;
99     boolean_t useselflg = 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.pstr =
163             (char *)ks_safe_strdup(optarg);
164         break;
165     case 'i':
166         nselflg = B_TRUE;
167         nselector->ks_instance.pstr =
168             (char *)ks_safe_strdup(optarg);
169         break;
170     case 'n':
171         nselflg = B_TRUE;
172         nselector->ks_name.pstr =
173             (char *)ks_safe_strdup(optarg);
174         break;
175     case 's':
176         nselflg = B_TRUE;
177         nselector->ks_statistic.pstr =
178             (char *)ks_safe_strdup(optarg);
179         break;
180     case 'c':
181         g_ks_class.pstr =
182             (char *)ks_safe_strdup(optarg);
183         break;
184     default:
185         errflg = B_TRUE;
186         break;
187     }
188
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     }
194
195     if (errflg) {
196         usage();
197         exit(2);
198     }
199
200     argc -= optind;
201     argv += optind;
202
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         }
221
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                 }
232
233                 if (*q != '\0') {
234                     switch (m) {
235                     case 1:
236                         uselector->ks_module.pstr =
237                             (char *)ks_safe_strdup(q);
238                         break;
239                     case 2:
240                         uselector->ks_instance.pstr =
241                             (char *)ks_safe_strdup(q);
242                         break;
243                     case 3:
244                         uselector->ks_name.pstr =
245                             (char *)ks_safe_strdup(q);
246                         break;
247                     case 4:
248                         uselector->ks_statistic.pstr =
249                             (char *)ks_safe_strdup(q);
250                         break;
251                     default:
252                         assert(B_FALSE);
253                     }
254                 }
255             }
256
257             if (m < 4) {
258                 free(uselector);

```

```

260         usage();
261         exit(2);
262     }

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

287 /*
288  * Check if we founded a named selector on the cmdline.
289  */
290 if (useselfg) {
291     if (nselflg) {
292         (void) fprintf(stderr, gettext(
293             "[module[:instance[:name[:statistic]]]] and "
294             "module:instance:name:statistic and "
295             "-m -i -n -s are mutually exclusive"));
296         usage();
297         exit(2);
298     } else {
299         free(nselector);
300     }
301 } else {
302     list_insert_tail(&selector_list, nselector);
303 }

304 assert(!list_is_empty(&selector_list));

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

309 while ((kc = kstat_open()) == NULL) {
310     if (errno == EAGAIN) {
311         (void) poll(NULL, 0, 200);
312     } else {
313         perror("kstat_open");
314         exit(3);
315     }
316 }

```

```

318     if (count > 1) {
319         if (signal(SIGCONT, cont_handler) == SIG_ERR) {
320             (void) fprintf(stderr, gettext(
321                 "signal failed"));
322             exit(3);
323         }
324     }

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

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

333         if (interval && count) {
334             ks_sleep_until(&start_n, period_n, infinite_cycles,
335                 &caught_cont);
336             (void) kstat_chain_update(kc);
337             (void) putchar('\n');
338         }
339     }

341     (void) kstat_close(kc);

343     return (g_matched);
344 }

346 /*
347  * Print usage.
348  */
349 static void
350 usage(void)
351 {
352     (void) fprintf(stderr, gettext(
353         "Usage:\n"
354         "kstat [ -Cjlpq ] [ -T d|u ] [ -c class ]\n"
355         "          [ -m module ] [ -i instance ] [ -n name ] [ -s statistic ]\n"
356         "          [ interval [ count ] ]\n"
357         "kstat [ -Cjlpq ] [ -T d|u ] [ -c class ]\n"
358         "          [ module[:instance[:name[:statistic]]] ... ]\n"
359         "          [ interval [ count ] ]\n"));
360 }

```

unchanged_portion_omitted

```

*****
9157 Wed Mar 13 17:08:30 2013
new/usr/src/man/man1m/kstat.1m
3623 kstat must accept partial stat specification
*****
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 \"Jan 9, 2013\"
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
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]
20 [\fImodule\fR[:\fIinstance\fR[:\fIname\fR[:\fIstatistic\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, or a regular expression
63 enclosed in '/' characters.
64 .sp
65 .ne 2
66 .na
67 \fB\C\fR
68 .ad
69 .RS 16n
70 Displays output in parseable format with a colon as separator.
71 .RE
73 .sp
74 .ne 2
75 .na
76 \fB-c\fR \fIclass\fR
77 .ad
78 .RS 16n
79 Displays only kstats that match the specified class. \fIclass\fR is a
80 kernel-defined string which classifies the "type" of the kstat.
81 .RE
83 .sp
84 .ne 2
85 .na
86 \fB-i\fR \fIinstance\fR
87 .ad
88 .RS 16n
89 Displays only kstats that match the specified instance.
90 .RE
92 .sp
93 .ne 2
94 .na
95 \fB-j\fR
96 .ad
97 .RS 16n
98 Displays output in JSON format.
99 .RE
101 .sp
102 .ne 2
103 .na
104 \fB-l\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-m\fR \fImodule\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-n\fR \fIname\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 or a regular expression enclosed by '/'
183 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_[0-9]+min$'\fR

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

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

240 cpu_stat:0:cpu_stat0:intr      29682330
241 cpu_stat:0:cpu_stat0:intrblk   87
242 cpu_stat:0:cpu_stat0:intrthread 15054222
243 cpu_stat:1:cpu_stat1:intr      426073
244 cpu_stat:1:cpu_stat1:intrblk   51
245 cpu_stat:1:cpu_stat1:intrthread 289668
246 cpu_stat:2:cpu_stat2:intr      134160
247 cpu_stat:2:cpu_stat2:intrblk   0
248 cpu_stat:2:cpu_stat2:intrthread 131
249 cpu_stat:3:cpu_stat3:intr      196566
250 cpu_stat:3:cpu_stat3:intrblk   30
251 cpu_stat:3:cpu_stat3:intrthread 59626
252 .fi
253 .in -2
254 .sp

256 .LP
257 \fBExample 3 \fRUsing the \fBkstat\fR Command
258 .sp

```

```

259 .in +2
260 .nf
261 example$ \fBkstat -p :::state '^^:avenrun*'\fR
262 example$ \fBkstat -p :::state ^^:^avenrun/\fR

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

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

285 unix:0:system_misc:avenrun_15min      15
286 unix:0:system_misc:avenrun_1min 11
287 unix:0:system_misc:avenrun_5min 21

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

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

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

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

```

```

325 .fi
326 .in -2
327 .sp

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

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

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

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

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

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

```