

new/usr/src/cmd/cpc/common/cpustat.c

```
*****  
2108 Tue Feb 3 05:15:47 2015  
new/usr/src/cmd/cpc/common/cpustat.c  
1100 cpustat usage message is incorrect  
*****  
_____ unchanged_portion_omitted _____  
107 static int cpustat(void);  
108 static int get_chipid(kstat_ctl_t *kc, processorid_t cpuid);  
109 static void *soaker(void *arg);  
  
112 #if !defined(TEXT_DOMAIN)  
113 #define TEXT_DOMAIN "SYS_TEST"  
114 #endif  
  
116 int  
117 main(int argc, char *argv[]) {  
118 {  
119     struct options *opts = &__options;  
120     int c, errcnt = 0, ret;  
121     cpc_setgrp_t *sgrp;  
122     char *errstr;  
123     double period;  
124     char *endp;  
125     struct rlimit rl;  
  
127     (void) setlocale(LC_ALL, "");  
128     (void) textdomain(TEXT_DOMAIN);  
  
130     if ((opts->pgmname = strrchr(argv[0], '/')) == NULL)  
131         opts->pgmname = argv[0];  
132     else  
133         opts->pgmname++;  
  
135     /* Make sure we can open enough files */  
136     rl.rlim_max = rl.rlim_cur = RLIM_INFINITY;  
137     if (setrlimit(RLIMIT_NOFILE, &rl) != 0) {  
138         errstr = strerror(errno);  
139         (void) fprintf(stderr,  
140             gettext("%s: setrlimit failed - %s\n"),  
141             opts->pgmname, errstr);  
142     }  
  
144     if ((cpc = cpc_open(CPC_VER_CURRENT)) == NULL) {  
145         errstr = strerror(errno);  
146         (void) fprintf(stderr, gettext("%s: cannot access performance "  
147             "counters - %s\n"), opts->pgmname, errstr);  
148         return (1);  
149     }  
  
151     (void) cpc_seterrhndlr(cpc, cpustat_errfn);  
152     strtoset_errfn = cpustat_errfn;  
  
154     /*  
155      * Check to see if cpustat needs to be SMT-aware.  
156      */  
157     smt = smt_limited_cpc_hw(cpc);  
  
159     /*  
160      * Establish some defaults  
161      */  
162     opts->mseconds = 5000;  
163     opts->namples = UINT_MAX;  
164     opts->dotitle = 1;  
165     if ((opts->master = cpc_setgrp_new(cpc, smt)) == NULL) {
```

1

new/usr/src/cmd/cpc/common/cpustat.c

```
166     (void) fprintf(stderr, gettext("%s: out of heap\n"),  
167                     opts->pgmname);  
168     return (1);  
169 }  
  
171 while ((c = getopt(argc, argv, "D:c:hnt:sp:")) != EOF && errcnt == 0)  
172     switch (c) {  
173         case 'D':  
174             /* enable debugging */  
175             opts->debug++;  
176             break;  
177         case 'c':  
178             /* specify statistics */  
179             if ((sgrp = cpc_setgrp_newset(opts->master,  
180                 optarg, &errcnt)) != NULL)  
181                 opts->master = sgrp;  
182             break;  
183         case 'n':  
184             /* no titles */  
185             opts->dotitle = 0;  
186             break;  
187         case 'p':  
188             /* periodic behavior */  
189             opts->doperiod = 1;  
190             period = strtod(optarg, &endp);  
191             if (*endp != '\0') {  
192                 (void) fprintf(stderr, gettext("%s: invalid "  
193                         "parameter \"%s\"\n"), opts->pgmname,  
194                         optarg);  
195                 errcnt++;  
196             }  
197             break;  
198         case 's':  
199             /* run soaker thread */  
200             opts->dosoaker = 1;  
201             break;  
202         case 't':  
203             /* print %tick */  
204             opts->dotick = 1;  
205             break;  
206         case 'T':  
207             if (optarg) {  
208                 if (*optarg == 'u')  
209                     timestamp_fmt = UPDATE;  
210                 else if (*optarg == 'd')  
211                     timestamp_fmt = DDATE;  
212                 else  
213                     errcnt++;  
214             }  
215             break;  
216         case 'h':  
217             /* help */  
218             opts->dohelp = 1;  
219             break;  
220         case '?':  
221             default:  
222                 errcnt++;  
223             break;  
224 }  
225 switch (argc - optind) {  
226 case 0:  
227     break;  
228 case 2:  
229     opts->namples = strtol(argv[optind + 1], &endp, 10);  
230     if (*endp != '\0') {  
231         (void) fprintf(stderr,  
232             gettext("%s: invalid argument \"%s\"\n"),  
233             opts->pgmname, argv[optind + 1]);  
234         errcnt++;  
235     }  
236     break;
```

2

```

232         }
233         /*FALLTHROUGH*/
234     case 1:
235         opts->mseconds = (uint_t)(strtod(argv[optind], &endp) * 1000.0);
236         if (*endp != '\0') {
237             (void) fprintf(stderr,
238                         gettext("%s: invalid argument \"%s\"\n"),
239                         argv[optind]);
240             errcnt++;
241         }
242         break;
243     default:
244         errcnt++;
245         break;
246     }

248     if (opts->namples == 0 || opts->mseconds == 0)
249         errcnt++;

251     if (errcnt != 0 || opts->dohelp ||
252         (opts->nsets = cpc_setgrp_numsets(opts->master)) == 0) {
253         (void) fprintf(opts->dohelp ? stdout : stderr, gettext(
254             "Usage:\n%s -c spec [-c spec]... [-p period] [-T u|d]\n"
255             "\t\t[-sntD] [interval [count]]\n"
256             "\t-c spec\t specify processor events to be monitored\n"
257             "Usage:\n%s [-c events] [-p period] [-nstD] "
258             "[[-T d/u] [interval [count]]]\n"
259             "\t-c events specify processor events to be monitored\n"
260             "\t-n\t suppress titles\n"
261             "\t-p\t period cycle through event list periodically\n"
262             "\t-s\t run user soaker thread for system-only events\n"
263             "\t-t\t include %s register\n"
264             "\t-d|ut\t Display a timestamp in date (d) or unix "
265             "time_t (u)\n"
266             "\t-D\t enable debug mode\n"
267             "\t-h\t print extended usage information\n"
268             "\tUse cputrack(1) to monitor per-process statistics.\n"),
269             argv[0], argv[0];
270         opts->pgmname, CPC_TICKREG_NAME);
271         if (opts->dohelp) {
272             (void) putchar('\n');
273             (void) capabilities(cpc, stdout);
274             exit(0);
275         }
276         exit(2);
277     }

278     /*
279      * If the user requested periodic behavior, calculate the rest time
280      * between cycles.
281      */
282     if (opts->doperiod) {
283         opts->mseconds_rest = (uint_t)((period * 1000.0) -
284             (opts->mseconds * opts->nsets));
285         if ((int)opts->mseconds_rest < 0)
286             opts->mseconds_rest = 0;
287         if (opts->namples != UINT_MAX)
288             opts->namples *= opts->nsets;
289     }

290     cpc_setgrp_reset(opts->master);
291     (void) setvbuf(stdout, NULL, _IOLBF, 0);

292     /*
293      * If no system-mode only sets were created, no soaker threads will be
294      * needed.
295      */

```

```

295     if (opts->dosoaker == 1 && cpc_setgrp_has_sysonly(opts->master) == 0)
296         opts->dosoaker = 0;
297
298     ret = cpustat();
299
300     (void) cpc_close(cpc);
301
302     return (ret);
303 }
```

unchanged portion omitted