```
**********************************************************
   14853 Thu Dec  6 15:01:42 2012
new/usr/src/cmd/psrinfo/psrinfo.c
3396 new psrinfo does not print socket type
Reviewed by: Alek Pinchuk <alek.pinchuk@nexenta.com>
**********************************************************
```

```
   1 /*
   2  * This file and its contents are supplied under the terms of the
   3  * Common Development and Distribution License ("CDDL"), version 1.0.
   4  * You may only use this file in accordance with the terms of version
   5  * 1.0 of the CDDL.
   6  *
   7  * A full copy of the text of the CDDL should have accompanied this
   8  * source.  A copy of the CDDL is also available via the Internet at
   9  * http://www.illumos.org/license/CDDL.
  10  */

  12 /*
  13  * Copyright (c) 2012 DEY Storage Systems, Inc.  All rights reserved.
  14  * Copyright 2012 Nexenta Systems, Inc.  All rights reserved.
  15  */

  17 /*
  18  * This implements psrinfo(1M), a utility to report various information
  19  * about processors, cores, and threads (virtual cpus).  This is mostly
  20  * intended for human consumption - this utility doesn't do much more than
  21  * simply process kstats for human readability.
  22  *
  23  * All the relevant kstats are in the cpu_info kstat module.
  24  */

  26 #include <stdio.h>
  27 #include <stdlib.h>
  28 #include <unistd.h>
  29 #include <string.h>
  30 #include <kstat.h>
  31 #include <libintl.h>
  32 #include <locale.h>
  33 #include <libgen.h>
  34 #include <ctype.h>
  35 #include <errno.h>

  37 #define _(x)    gettext(x)
  38 #if XGETTEXT
  39 /* These CPU states are here for benefit of xgettext */
  40 _("on-line")
  41 _("off-line")
  42 _("faulted")
  43 _("powered-off")
  44 _("no-intr")
  45 _("spare")
  46 _("unknown")
  47 #endif

  49 /*
  50  * We deal with sorted linked lists, where the sort key is usually the
  51  * cpu id, core id, or chip id.  We generalize this with simple node.
  52  */
  53 struct link {
  54         long            l_id;
  55         struct link     *l_next;
  56         void            *l_ptr;
  57 };
```

*unchanged_portion_omitted_*

```
  82 struct vcpu {
```



```
  83         struct link     v_link;

  85         struct link     v_link_core;
  86         struct link     v_link_pchip;

  88         int             v_doit;

  90         struct pchip    *v_pchip;
  91         struct core     *v_core;

  93         char            *v_state;
  94         long            v_state_begin;
  95         char            *v_cpu_type;
  96         char            *v_fpu_type;
  97         long            v_clock_mhz;
  98         long            v_pchip_id;     /* 1 per socket */
  99         char            *v_impl;
 100         char            *v_brand;
 101         char            *v_socket;
 102         long            v_core_id;      /* n per chip_id */
 103 };
```

*unchanged_portion_omitted_*

```
 225 static void
 226 print_vp(int nspec)
 227 {
 228         struct pchip *chip;
 229         struct core *core;
 230         struct vcpu *vcpu;
 231         struct link *l1, *l2;
 232         int len;
 233         for (l1 = pchips; l1; l1 = l1->l_next) {

 235                 chip = l1->l_ptr;

 237                 if ((nspec != 0) && (chip->p_doit == 0))
 238                         continue;

 240                 vcpu = chip->p_vcpus->l_ptr;

 242                 /*
 243                  * Note that some of the way these strings are broken up are
 244                  * to accommodate the legacy translations so that we won't
 245                  * have to retranslate for this utility.
 246                  */
 247                 if ((chip->p_ncore == 1) || (chip->p_ncore == chip->p_nvcpu)) {
 248                         (void) printf(_("%s has %d virtual %s"),
 249                             _("The physical processor"),
 250                             chip->p_nvcpu,
 251                             chip->p_nvcpu > 1 ?
 252                             _("processors") :
 253                             _("processor"));
 254                 } else {
 255                         (void) printf(_("%s has %d %s and %d virtual %s"),
 256                             _("The physical processor"),
 257                             chip->p_ncore, _("cores"),
 258                             chip->p_nvcpu,
 259                             chip->p_nvcpu > 1 ?
 260                             _("processors") : _("processor"));
 261                 }

 263                 print_links(chip->p_vcpus);
 264                 (void) putchar('\n');

 266                 if ((chip->p_ncore == 1) || (chip->p_ncore == chip->p_nvcpu)) {
 267                         if (strlen(vcpu->v_impl)) {
```

```
268                             (void) printf("  %s\n", vcpu->v_impl);
269                     }
270                     if (((len = strlen(vcpu->v_brand)) != 0) &&
271                         (strncmp(vcpu->v_brand, vcpu->v_impl, len) != 0))
272                             (void) printf("\t%s", vcpu->v_brand);
273                     if (strcmp(vcpu->v_socket, "Unknown") != 0)
274                             (void) printf("\t[ %s: %s ]", _("Socket"),
275                                 vcpu->v_socket);
276                     (void) putchar('\n');
277             } else {
278                     for (l2 = chip->p_cores; l2; l2 = l2->l_next) {
279                             core = l2->l_ptr;
280                             (void) printf(_("  %s has %d virtual %s"),
281                                 _("The core"),
282                                 core->c_nvcpu,
283                                 chip->p_nvcpu > 1 ?
284                                 _("processors") : _("processor"));
285                             print_links(core->c_vcpus);
286                             (void) putchar('\n');
287                     }
288                     if (strlen(vcpu->v_impl)) {
289                             (void) printf("    %s\n", vcpu->v_impl);
290                     }
291                     if (((len = strlen(vcpu->v_brand)) != 0) &&
292                         (strncmp(vcpu->v_brand, vcpu->v_impl, len) != 0))
293                             (void) printf("      %s\n", vcpu->v_brand);
294             }
295         }
296 }
```

***unchanged_portion_omitted_***

```
435 int
436 main(int argc, char **argv)
437 {
438         kstat_ctl_t     *kc;
439         kstat_t         *ksp;
440         kstat_named_t   *knp;
441         struct vcpu     *vc;
442         struct core     *core;
443         struct pchip    *chip;
444         struct link     **ins;
445         char            *s;
446         int             nspec;
447         int             optc;
448         int             opt_s = 0;
449         int             opt_p = 0;
450         int             opt_v = 0;
451         int             ex = 0;

453         cmdname = basename(argv[0]);


456         (void) setlocale(LC_ALL, "");
457 #if !defined(TEXT_DOMAIN)
458 #define TEXT_DOMAIN     "SYS_TEST"
459 #endif
460         (void) textdomain(TEXT_DOMAIN);

462         /* collect the kstats */
463         if ((kc = kstat_open()) == NULL)
464                 die(_("kstat_open() failed"));

466         if ((ksp = kstat_lookup(kc, "cpu_info", -1, NULL)) == NULL)
467                 die(_("kstat_lookup() failed"));

469         for (ksp = kc->kc_chain; ksp; ksp = ksp->ks_next) {
```



```
471                 if (strcmp(ksp->ks_module, "cpu_info") != 0)
472                         continue;
473                 if (kstat_read(kc, ksp, NULL) == NULL)
474                         die(_("kstat_read() failed"));

476                 vc = find_link(&vcpus, ksp->ks_instance, &ins);
477                 if (vc == NULL) {
478                         vc = zalloc(sizeof (struct vcpu));
479                         vc->v_link.l_id = ksp->ks_instance;
480                         vc->v_link_core.l_id = ksp->ks_instance;
481                         vc->v_link_pchip.l_id = ksp->ks_instance;
482                         vc->v_link.l_ptr = vc;
483                         vc->v_link_core.l_ptr = vc;
484                         vc->v_link_pchip.l_ptr = vc;
485                         ins_link(ins, &vc->v_link);
486                 }

488                 if ((knp = kstat_data_lookup(ksp, "state")) != NULL) {
489                         vc->v_state = mystrdup(knp->value.c);
490                 } else {
491                         vc->v_state = "unknown";
492                 }

494                 if ((knp = kstat_data_lookup(ksp, "cpu_type")) != NULL) {
495                         vc->v_cpu_type = mystrdup(knp->value.c);
496                 }
497                 if ((knp = kstat_data_lookup(ksp, "fpu_type")) != NULL) {
498                         vc->v_fpu_type = mystrdup(knp->value.c);
499                 }

501                 if ((knp = kstat_data_lookup(ksp, "state_begin")) != NULL) {
502                         vc->v_state_begin = knp->value.l;
503                 }

505                 if ((knp = kstat_data_lookup(ksp, "clock_MHz")) != NULL) {
506                         vc->v_clock_mhz = knp->value.l;
507                 }

509                 if ((knp = kstat_data_lookup(ksp, "brand")) == NULL) {
510                         vc->v_brand = _("(unknown)");
511                 } else {
512                         vc->v_brand = mystrdup(knp->value.str.addr.ptr);
513                 }

515                 if ((knp = kstat_data_lookup(ksp, "socket_type")) == NULL) {
516                         vc->v_socket = "Unknown";
517                 } else {
518                         vc->v_socket = mystrdup(knp->value.str.addr.ptr);
519                 }

521                 if ((knp = kstat_data_lookup(ksp, "implementation")) == NULL) {
522                         vc->v_impl = _("(unknown)");
523                 } else {
524                         vc->v_impl = mystrdup(knp->value.str.addr.ptr);
525                 }
526                 /*
527                  * Legacy code removed the chipid and cpuid fields... we
528                  * do the same for compatibility.  Note that the original
529                  * pattern is a bit strange, and we have to emulate this because
530                  * on SPARC we *do* emit these.  The original pattern we are
531                  * emulating is: $impl =~ s/(cpuid|chipid)\s*\w+\s+//;
532                  */
533                 if ((s = strstr(vc->v_impl, "chipid")) != NULL) {
534                         char *x = s + strlen("chipid");
535                         while (isspace(*x))
```

```
536                             x++;
537                     if ((!isalnum(*x)) && (*x != '_'))
538                             goto nochipid;
539                     while (isalnum(*x) || (*x == '_'))
540                             x++;
541                     if (!isspace(*x))
542                             goto nochipid;
543                     while (isspace(*x))
544                             x++;
545                     (void) strcpy(s, x);
546             }
547 nochipid:
548             if ((s = strstr(vc->v_impl, "cpuid")) != NULL) {
549                     char *x = s + strlen("cpuid");
550                     while (isspace(*x))
551                             x++;
552                     if ((!isalnum(*x)) && (*x != '_'))
553                             goto nocpuid;
554                     while (isalnum(*x) || (*x == '_'))
555                             x++;
556                     if (!isspace(*x))
557                             goto nocpuid;
558                     while (isspace(*x))
559                             x++;
560                     (void) strcpy(s, x);
561             }
562 nocpuid:

564             if ((knp = kstat_data_lookup(ksp, "chip_id")) != NULL)
565                     vc->v_pchip_id = knp->value.l;
566             chip = find_link(&pchips, vc->v_pchip_id, &ins);
567             if (chip == NULL) {
568                     chip = zalloc(sizeof (struct pchip));
569                     chip->p_link.l_id = vc->v_pchip_id;
570                     chip->p_link.l_ptr = chip;
571                     ins_link(ins, &chip->p_link);
572             }
573             vc->v_pchip = chip;

575             if ((knp = kstat_data_lookup(ksp, "core_id")) != NULL)
576                     vc->v_core_id = knp->value.l;
577             core = find_link(&cores, vc->v_core_id, &ins);
578             if (core == NULL) {
579                     core = zalloc(sizeof (struct core));
580                     core->c_link.l_id = vc->v_core_id;
581                     core->c_link.l_ptr = core;
582                     core->c_link_pchip.l_id = vc->v_core_id;
583                     core->c_link_pchip.l_ptr = core;
584                     core->c_pchip = chip;
585                     ins_link(ins, &core->c_link);
586                     chip->p_ncore++;
587                     (void) find_link(&chip->p_cores, core->c_link.l_id,
588                         &ins);
589                     ins_link(ins, &core->c_link_pchip);
590             }
591             vc->v_core = core;




595             /* now put other linkages in place */
596             (void) find_link(&chip->p_vcpus, vc->v_link.l_id, &ins);
597             ins_link(ins, &vc->v_link_pchip);
598             chip->p_nvcpu++;

600             (void) find_link(&core->c_vcpus, vc->v_link.l_id, &ins);
601             ins_link(ins, &vc->v_link_core);
```



```
602                     core->c_nvcpu++;
603             }

605             (void) kstat_close(kc);

607             nspec = 0;

609             while ((optc = getopt(argc, argv, "pvs")) != EOF) {
610                     switch (optc) {
611                     case 's':
612                             opt_s = 1;
613                             break;
614                     case 'p':
615                             opt_p = 1;
616                             break;
617                     case 'v':
618                             opt_v = 1;
619                             break;
620                     default:
621                             usage(NULL);
622                     }
623             }

625             while (optind < argc) {
626                     long id;
627                     char *eptr;
628                     struct link *l;
629                     id = strtol(argv[optind], &eptr, 10);
630                     l = find_link(&vcpus, id, NULL);
631                     if ((*eptr != '\0') || (l == NULL)) {
632                             (void) fprintf(stderr,
633                                 _("%s: processor %s: Invalid argument\n"),
634                                 cmdname, argv[optind]);
635                             ex = 2;
636                     } else {
637                             ((struct vcpu *)l->l_ptr)->v_doit = 1;
638                             ((struct vcpu *)l->l_ptr)->v_pchip->p_doit = 1;
639                             ((struct vcpu *)l->l_ptr)->v_core->c_doit = 1;
640                     }
641                     nspec++;
642                     optind++;
643             }

645             if (opt_s && opt_v) {
646                     usage(_("options -s and -v are mutually exclusive"));
647             }
648             if (opt_s && nspec != 1) {
649                     usage(_("must specify exactly one processor if -s used"));
650             }
651             if (opt_v && opt_p) {
652                     print_vp(nspec);
653             } else if (opt_s && opt_p) {
654                     print_ps();
655             } else if (opt_p) {
656                     print_p(nspec);
657             } else if (opt_v) {
658                     print_v(nspec);
659             } else if (opt_s) {
660                     print_s();
661             } else {
662                     print_normal(nspec);
663             }

665             return (ex);
666 }
```

_____***unchanged_portion_omitted_***