```
**********************************************************
   36652 Thu Mar  5 12:02:13 2015
new/usr/src/cmd/stat/kstat/kstat.c
4740 kstat(1M) prints wrong snaptime
**********************************************************
______unchanged_portion_omitted_

 581 /*
 582  * Iterate over all kernel statistics and save matches.
 583  */
 584 static void
 585 ks_instances_read(kstat_ctl_t *kc)
 586 {
 587         kstat_raw_reader_t save_raw = NULL;
 588         kid_t              id;
 589         ks_selector_t   *selector;
 590         ks_instance_t   *ksi;
 591         ks_instance_t   *tmp;
 592         kstat_t         *kp;
 593         boolean_t       skip;

 595         for (kp = kc->kc_chain; kp != NULL; kp = kp->ks_next) {
 596                 /* Don't bother storing the kstat headers */
 597                 if (strncmp(kp->ks_name, "kstat_", 6) == 0) {
 598                         continue;
 599                 }

 601                 /* Don't bother storing raw stats we don't understand */
 602                 if (kp->ks_type == KSTAT_TYPE_RAW) {
 603                         save_raw = lookup_raw_kstat_fn(kp->ks_module,
 604                             kp->ks_name);
 605                         if (save_raw == NULL) {
 606 #ifdef REPORT_UNKNOWN
 607                                 (void) fprintf(stderr,
 608                                     "Unknown kstat type %s:%d:%s - "
 609                                     "%d of size %d\n", kp->ks_module,
 610                                     kp->ks_instance, kp->ks_name,
 611                                     kp->ks_ndata, kp->ks_data_size);
 612 #endif
 613                                 continue;
 614                         }
 615                 }

 617                 /*
 618                  * Iterate over the list of selectors and skip
 619                  * instances we dont want. We filter for statistics
 620                  * later, as we dont know them yet.
 621                  */
 622                 skip = B_TRUE;
 623                 selector = list_head(&selector_list);
 624                 while (selector != NULL) {
 625                         if (ks_match(kp->ks_module, &selector->ks_module) &&
 626                             ks_match(kp->ks_name, &selector->ks_name)) {
 627                                 skip = B_FALSE;
 628                                 break;
 629                         }
 630                         selector = list_next(&selector_list, selector);
 631                 }

 633                 if (skip) {
 634                         continue;
 635                 }

 637                 /*
 638                  * Allocate a new instance and fill in the values
 639                  * we know so far.
```



```
 640                  */
 641                 ksi = (ks_instance_t *)malloc(sizeof (ks_instance_t));
 642                 if (ksi == NULL) {
 643                         perror("malloc");
 644                         exit(3);
 645                 }

 647                 list_link_init(&ksi->ks_next);

 649                 (void) strlcpy(ksi->ks_module, kp->ks_module, KSTAT_STRLEN);
 650                 (void) strlcpy(ksi->ks_name, kp->ks_name, KSTAT_STRLEN);
 651                 (void) strlcpy(ksi->ks_class, kp->ks_class, KSTAT_STRLEN);

 653                 ksi->ks_instance = kp->ks_instance;
 654                 ksi->ks_snaptime = kp->ks_snaptime;
 655                 ksi->ks_type = kp->ks_type;

 657                 list_create(&ksi->ks_nvlist, sizeof (ks_nvpair_t),
 658                     offsetof(ks_nvpair_t, nv_next));

 660                 SAVE_HRTIME_X(ksi, "crtime", kp->ks_crtime);
 661                 SAVE_HRTIME_X(ksi, "snaptime", kp->ks_snaptime);
 662                 if (g_pflg) {
 663                         SAVE_STRING_X(ksi, "class", kp->ks_class);
 664                 }

 660                 /* Insert this instance into a sorted list */
 661                 tmp = list_head(&instances_list);
 662                 while (tmp != NULL && compare_instances(ksi, tmp) > 0)
 663                         tmp = list_next(&instances_list, tmp);

 665                 list_insert_before(&instances_list, tmp, ksi);

 667                 /* Read the actual statistics */
 668                 id = kstat_read(kc, kp, NULL);
 669                 if (id == -1) {
 670 #ifdef REPORT_UNKNOWN
 671                         perror("kstat_read");
 672 #endif
 673                         continue;
 674                 }

 676                 SAVE_HRTIME_X(ksi, "crtime", kp->ks_crtime);
 677                 SAVE_HRTIME_X(ksi, "snaptime", kp->ks_snaptime);
 678                 if (g_pflg) {
 679                         SAVE_STRING_X(ksi, "class", kp->ks_class);
 680                 }

 682 #endif /* ! codereview */
 683                 switch (kp->ks_type) {
 684                 case KSTAT_TYPE_RAW:
 685                         save_raw(kp, ksi);
 686                         break;
 687                 case KSTAT_TYPE_NAMED:
 688                         save_named(kp, ksi);
 689                         break;
 690                 case KSTAT_TYPE_INTR:
 691                         save_intr(kp, ksi);
 692                         break;
 693                 case KSTAT_TYPE_IO:
 694                         save_io(kp, ksi);
 695                         break;
 696                 case KSTAT_TYPE_TIMER:
 697                         save_timer(kp, ksi);
 698                         break;
 699                 default:
```

```
700                         assert(B_FALSE); /* Invalid type */
701                         break;
702                 }
703         }
704 }

706 /*
707  * Print the value of a name-value pair.
708  */
709 static void
710 ks_value_print(ks_nvpair_t *nvpair)
711 {
712         switch (nvpair->data_type) {
713         case KSTAT_DATA_CHAR:
714                 (void) fprintf(stdout, "%s", nvpair->value.c);
715                 break;
716         case KSTAT_DATA_INT32:
717                 (void) fprintf(stdout, "%d", nvpair->value.i32);
718                 break;
719         case KSTAT_DATA_UINT32:
720                 (void) fprintf(stdout, "%u", nvpair->value.ui32);
721                 break;
722         case KSTAT_DATA_INT64:
723                 (void) fprintf(stdout, "%lld", nvpair->value.i64);
724                 break;
725         case KSTAT_DATA_UINT64:
726                 (void) fprintf(stdout, "%llu", nvpair->value.ui64);
727                 break;
728         case KSTAT_DATA_STRING:
729                 (void) fprintf(stdout, "%s", KSTAT_NAMED_STR_PTR(nvpair));
730                 break;
731         case KSTAT_DATA_HRTIME:
732                 if (nvpair->value.ui64 == 0)
733                         (void) fprintf(stdout, "0");
734                 else
735                         (void) fprintf(stdout, "%.9f",
736                             nvpair->value.ui64 / 1000000000.0);
737                 break;
738         default:
739                 assert(B_FALSE);
740         }
741 }

743 /*
744  * Print a single instance.
745  */
746 static void
747 ks_instance_print(ks_instance_t *ksi, ks_nvpair_t *nvpair)
748 {
749         if (g_headerflg) {
750                 if (!g_pflg) {
751                         (void) fprintf(stdout, DFLT_FMT,
752                             ksi->ks_module, ksi->ks_instance,
753                             ksi->ks_name, ksi->ks_class);
754                 }
755                 g_headerflg = B_FALSE;
756         }

758         if (g_pflg) {
759                 (void) fprintf(stdout, KS_PFMT,
760                     ksi->ks_module, ksi->ks_instance,
761                     ksi->ks_name, nvpair->name);
762                 if (!g_lflg) {
763                         (void) putchar(g_cflg ? ':': '\t');
764                         ks_value_print(nvpair);
765                 }
```



```
766         } else {
767                 (void) fprintf(stdout, KS_DFMT, nvpair->name);
768                 ks_value_print(nvpair);
769         }

771         (void) putchar('\n');
772 }

774 /*
775  * Print a single instance in JSON format.
776  */
777 static void
778 ks_instance_print_json(ks_instance_t *ksi, ks_nvpair_t *nvpair)
779 {
780         if (g_headerflg) {
781                 (void) fprintf(stdout, JSON_FMT,
782                     ksi->ks_module, ksi->ks_instance,
783                     ksi->ks_name, ksi->ks_class,
784                     ksi->ks_type);

786                 if (ksi->ks_snaptime == 0)
787                         (void) fprintf(stdout, "\t\"snaptime\": 0,\n");
788                 else
789                         (void) fprintf(stdout, "\t\"snaptime\": %.9f,\n",
790                             ksi->ks_snaptime / 1000000000.0);

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

794                 g_headerflg = B_FALSE;
795         }

797         (void) fprintf(stdout, KS_JFMT, nvpair->name);
798         if (nvpair->data_type == KSTAT_DATA_STRING) {
799                 (void) putchar('\"');
800                 ks_value_print(nvpair);
801                 (void) putchar('\"');
802         } else {
803                 ks_value_print(nvpair);
804         }
805         if (nvpair != list_tail(&ksi->ks_nvlist))
806                 (void) putchar(',');

808         (void) putchar('\n');
809 }

811 /*
812  * Print all instances.
813  */
814 static void
815 ks_instances_print(void)
816 {
817         ks_selector_t   *selector;
818         ks_instance_t   *ksi, *ktmp;
819         ks_nvpair_t     *nvpair, *ntmp;
820         void            (*ks_print_fn)(ks_instance_t *, ks_nvpair_t *);
821         char            *ks_number;

823         if (g_timestamp_fmt != NODATE)
824                 print_timestamp(g_timestamp_fmt);

826         if (g_jflg) {
827                 ks_print_fn = &ks_instance_print_json;
828                 (void) putchar('[');
829         } else {
830                 ks_print_fn = &ks_instance_print;
831         }
```

```
833         /* Iterate over each selector */
834         selector = list_head(&selector_list);
835         while (selector != NULL) {

837                 /* Iterate over each instance */
838                 for (ksi = list_head(&instances_list); ksi != NULL;
839                     ksi = list_next(&instances_list, ksi)) {

841                         (void) asprintf(&ks_number, "%d", ksi->ks_instance);
842                         if (!(ks_match(ksi->ks_module, &selector->ks_module) &&
843                             ks_match(ksi->ks_name, &selector->ks_name) &&
844                             ks_match(ks_number, &selector->ks_instance) &&
845                             ks_match(ksi->ks_class, &g_ks_class))) {
846                                 free(ks_number);
847                                 continue;
848                         }

850                         free(ks_number);

852                         /* Finally iterate over each statistic */
853                         g_headerflg = B_TRUE;
854                         for (nvpair = list_head(&ksi->ks_nvlist);
855                             nvpair != NULL;
856                             nvpair = list_next(&ksi->ks_nvlist, nvpair)) {
857                                 if (!ks_match(nvpair->name,
858                                     &selector->ks_statistic))
859                                         continue;

861                                 g_matched = 0;
862                                 if (!g_qflg)
863                                         (*ks_print_fn)(ksi, nvpair);
864                         }

866                         if (!g_headerflg) {
867                                 if (g_jflg) {
868                                         (void) fprintf(stdout, "\t}\n}");
869                                         if (ksi != list_tail(&instances_list))
870                                                 (void) putchar(',');
871                                 } else if (!g_pflg) {
872                                         (void) putchar('\n');
873                                 }
874                         }
875                 }

877                 selector = list_next(&selector_list, selector);
878         }

880         if (g_jflg)
881                 (void) fprintf(stdout, "]\n");

883         (void) fflush(stdout);

885         /* Free the instances list */
886         ksi = list_head(&instances_list);
887         while (ksi != NULL) {
888                 nvpair = list_head(&ksi->ks_nvlist);
889                 while (nvpair != NULL) {
890                         ntmp = nvpair;
891                         nvpair = list_next(&ksi->ks_nvlist, nvpair);
892                         list_remove(&ksi->ks_nvlist, ntmp);
893                         if (ntmp->data_type == KSTAT_DATA_STRING)
894                                 free(ntmp->value.str.addr.ptr);
895                         free(ntmp);
896                 }
```



```
898                 ktmp = ksi;
899                 ksi = list_next(&instances_list, ksi);
900                 list_remove(&instances_list, ktmp);
901                 list_destroy(&ktmp->ks_nvlist);
902                 free(ktmp);
903         }
904 }

906 static void
907 save_cpu_stat(kstat_t *kp, ks_instance_t *ksi)
908 {
909         cpu_stat_t      *stat;
910         cpu_sysinfo_t   *sysinfo;
911         cpu_syswait_t   *syswait;
912         cpu_vminfo_t    *vminfo;

914         stat = (cpu_stat_t *)(kp->ks_data);
915         sysinfo = &stat->cpu_sysinfo;
916         syswait = &stat->cpu_syswait;
917         vminfo  = &stat->cpu_vminfo;

919         SAVE_UINT32_X(ksi, "idle", sysinfo->cpu[CPU_IDLE]);
920         SAVE_UINT32_X(ksi, "user", sysinfo->cpu[CPU_USER]);
921         SAVE_UINT32_X(ksi, "kernel", sysinfo->cpu[CPU_KERNEL]);
922         SAVE_UINT32_X(ksi, "wait", sysinfo->cpu[CPU_WAIT]);
923         SAVE_UINT32_X(ksi, "wait_io", sysinfo->wait[W_IO]);
924         SAVE_UINT32_X(ksi, "wait_swap", sysinfo->wait[W_SWAP]);
925         SAVE_UINT32_X(ksi, "wait_pio", sysinfo->wait[W_PIO]);
926         SAVE_UINT32(ksi, sysinfo, bread);
927         SAVE_UINT32(ksi, sysinfo, bwrite);
928         SAVE_UINT32(ksi, sysinfo, lread);
929         SAVE_UINT32(ksi, sysinfo, lwrite);
930         SAVE_UINT32(ksi, sysinfo, phread);
931         SAVE_UINT32(ksi, sysinfo, phwrite);
932         SAVE_UINT32(ksi, sysinfo, pswitch);
933         SAVE_UINT32(ksi, sysinfo, trap);
934         SAVE_UINT32(ksi, sysinfo, intr);
935         SAVE_UINT32(ksi, sysinfo, syscall);
936         SAVE_UINT32(ksi, sysinfo, sysread);
937         SAVE_UINT32(ksi, sysinfo, syswrite);
938         SAVE_UINT32(ksi, sysinfo, sysfork);
939         SAVE_UINT32(ksi, sysinfo, sysvfork);
940         SAVE_UINT32(ksi, sysinfo, sysexec);
941         SAVE_UINT32(ksi, sysinfo, readch);
942         SAVE_UINT32(ksi, sysinfo, writech);
943         SAVE_UINT32(ksi, sysinfo, rcvint);
944         SAVE_UINT32(ksi, sysinfo, xmtint);
945         SAVE_UINT32(ksi, sysinfo, mdmint);
946         SAVE_UINT32(ksi, sysinfo, rawch);
947         SAVE_UINT32(ksi, sysinfo, canch);
948         SAVE_UINT32(ksi, sysinfo, outch);
949         SAVE_UINT32(ksi, sysinfo, msg);
950         SAVE_UINT32(ksi, sysinfo, sema);
951         SAVE_UINT32(ksi, sysinfo, namei);
952         SAVE_UINT32(ksi, sysinfo, ufsiget);
953         SAVE_UINT32(ksi, sysinfo, ufsdirblk);
954         SAVE_UINT32(ksi, sysinfo, ufsipage);
955         SAVE_UINT32(ksi, sysinfo, ufsinopage);
956         SAVE_UINT32(ksi, sysinfo, inodeovf);
957         SAVE_UINT32(ksi, sysinfo, fileovf);
958         SAVE_UINT32(ksi, sysinfo, procovf);
959         SAVE_UINT32(ksi, sysinfo, intrthread);
960         SAVE_UINT32(ksi, sysinfo, intrblk);
961         SAVE_UINT32(ksi, sysinfo, idlethread);
962         SAVE_UINT32(ksi, sysinfo, inv_swtch);
963         SAVE_UINT32(ksi, sysinfo, nthreads);
```

```
 964         SAVE_UINT32(ksi, sysinfo, cpumigrate);
 965         SAVE_UINT32(ksi, sysinfo, xcalls);
 966         SAVE_UINT32(ksi, sysinfo, mutex_adenters);
 967         SAVE_UINT32(ksi, sysinfo, rw_rdfails);
 968         SAVE_UINT32(ksi, sysinfo, rw_wrfails);
 969         SAVE_UINT32(ksi, sysinfo, modload);
 970         SAVE_UINT32(ksi, sysinfo, modunload);
 971         SAVE_UINT32(ksi, sysinfo, bawrite);
 972 #ifdef  STATISTICS      /* see header file */
 973         SAVE_UINT32(ksi, sysinfo, rw_enters);
 974         SAVE_UINT32(ksi, sysinfo, win_uo_cnt);
 975         SAVE_UINT32(ksi, sysinfo, win_uu_cnt);
 976         SAVE_UINT32(ksi, sysinfo, win_so_cnt);
 977         SAVE_UINT32(ksi, sysinfo, win_su_cnt);
 978         SAVE_UINT32(ksi, sysinfo, win_suo_cnt);
 979 #endif

 981         SAVE_INT32(ksi, syswait, iowait);
 982         SAVE_INT32(ksi, syswait, swap);
 983         SAVE_INT32(ksi, syswait, physio);

 985         SAVE_UINT32(ksi, vminfo, pgrec);
 986         SAVE_UINT32(ksi, vminfo, pgfrec);
 987         SAVE_UINT32(ksi, vminfo, pgin);
 988         SAVE_UINT32(ksi, vminfo, pgpgin);
 989         SAVE_UINT32(ksi, vminfo, pgout);
 990         SAVE_UINT32(ksi, vminfo, pgpgout);
 991         SAVE_UINT32(ksi, vminfo, swapin);
 992         SAVE_UINT32(ksi, vminfo, pgswapin);
 993         SAVE_UINT32(ksi, vminfo, swapout);
 994         SAVE_UINT32(ksi, vminfo, pgswapout);
 995         SAVE_UINT32(ksi, vminfo, zfod);
 996         SAVE_UINT32(ksi, vminfo, dfree);
 997         SAVE_UINT32(ksi, vminfo, scan);
 998         SAVE_UINT32(ksi, vminfo, rev);
 999         SAVE_UINT32(ksi, vminfo, hat_fault);
1000         SAVE_UINT32(ksi, vminfo, as_fault);
1001         SAVE_UINT32(ksi, vminfo, maj_fault);
1002         SAVE_UINT32(ksi, vminfo, cow_fault);
1003         SAVE_UINT32(ksi, vminfo, prot_fault);
1004         SAVE_UINT32(ksi, vminfo, softlock);
1005         SAVE_UINT32(ksi, vminfo, kernel_asflt);
1006         SAVE_UINT32(ksi, vminfo, pgrrun);
1007         SAVE_UINT32(ksi, vminfo, execpgin);
1008         SAVE_UINT32(ksi, vminfo, execpgout);
1009         SAVE_UINT32(ksi, vminfo, execfree);
1010         SAVE_UINT32(ksi, vminfo, anonpgin);
1011         SAVE_UINT32(ksi, vminfo, anonpgout);
1012         SAVE_UINT32(ksi, vminfo, anonfree);
1013         SAVE_UINT32(ksi, vminfo, fspgin);
1014         SAVE_UINT32(ksi, vminfo, fspgout);
1015         SAVE_UINT32(ksi, vminfo, fsfree);
1016 }

1018 static void
1019 save_var(kstat_t *kp, ks_instance_t *ksi)
1020 {
1021         struct var      *var = (struct var *)(kp->ks_data);

1023         assert(kp->ks_data_size == sizeof (struct var));

1025         SAVE_INT32(ksi, var, v_buf);
1026         SAVE_INT32(ksi, var, v_call);
1027         SAVE_INT32(ksi, var, v_proc);
1028         SAVE_INT32(ksi, var, v_maxupttl);
1029         SAVE_INT32(ksi, var, v_nglobpris);
```



```
1030         SAVE_INT32(ksi, var, v_maxsyspri);
1031         SAVE_INT32(ksi, var, v_clist);
1032         SAVE_INT32(ksi, var, v_maxup);
1033         SAVE_INT32(ksi, var, v_hbuf);
1034         SAVE_INT32(ksi, var, v_hmask);
1035         SAVE_INT32(ksi, var, v_pbuf);
1036         SAVE_INT32(ksi, var, v_sptmap);
1037         SAVE_INT32(ksi, var, v_maxpmem);
1038         SAVE_INT32(ksi, var, v_autoup);
1039         SAVE_INT32(ksi, var, v_bufhwm);
1040 }

1042 static void
1043 save_ncstats(kstat_t *kp, ks_instance_t *ksi)
1044 {
1045         struct ncstats  *ncstats = (struct ncstats *)(kp->ks_data);

1047         assert(kp->ks_data_size == sizeof (struct ncstats));

1049         SAVE_INT32(ksi, ncstats, hits);
1050         SAVE_INT32(ksi, ncstats, misses);
1051         SAVE_INT32(ksi, ncstats, enters);
1052         SAVE_INT32(ksi, ncstats, dbl_enters);
1053         SAVE_INT32(ksi, ncstats, long_enter);
1054         SAVE_INT32(ksi, ncstats, long_look);
1055         SAVE_INT32(ksi, ncstats, move_to_front);
1056         SAVE_INT32(ksi, ncstats, purges);
1057 }

1059 static void
1060 save_sysinfo(kstat_t *kp, ks_instance_t *ksi)
1061 {
1062         sysinfo_t       *sysinfo = (sysinfo_t *)(kp->ks_data);

1064         assert(kp->ks_data_size == sizeof (sysinfo_t));

1066         SAVE_UINT32(ksi, sysinfo, updates);
1067         SAVE_UINT32(ksi, sysinfo, runque);
1068         SAVE_UINT32(ksi, sysinfo, runocc);
1069         SAVE_UINT32(ksi, sysinfo, swpque);
1070         SAVE_UINT32(ksi, sysinfo, swpocc);
1071         SAVE_UINT32(ksi, sysinfo, waiting);
1072 }

1074 static void
1075 save_vminfo(kstat_t *kp, ks_instance_t *ksi)
1076 {
1077         vminfo_t        *vminfo = (vminfo_t *)(kp->ks_data);

1079         assert(kp->ks_data_size == sizeof (vminfo_t));

1081         SAVE_UINT64(ksi, vminfo, freemem);
1082         SAVE_UINT64(ksi, vminfo, swap_resv);
1083         SAVE_UINT64(ksi, vminfo, swap_alloc);
1084         SAVE_UINT64(ksi, vminfo, swap_avail);
1085         SAVE_UINT64(ksi, vminfo, swap_free);
1086         SAVE_UINT64(ksi, vminfo, updates);
1087 }

1089 static void
1090 save_nfs(kstat_t *kp, ks_instance_t *ksi)
1091 {
1092         struct mntinfo_kstat *mntinfo = (struct mntinfo_kstat *)(kp->ks_data);

1094         assert(kp->ks_data_size == sizeof (struct mntinfo_kstat));
```

```
1096         SAVE_STRING(ksi, mntinfo, mik_proto);
1097         SAVE_UINT32(ksi, mntinfo, mik_vers);
1098         SAVE_UINT32(ksi, mntinfo, mik_flags);
1099         SAVE_UINT32(ksi, mntinfo, mik_secmod);
1100         SAVE_UINT32(ksi, mntinfo, mik_curread);
1101         SAVE_UINT32(ksi, mntinfo, mik_curwrite);
1102         SAVE_INT32(ksi, mntinfo, mik_timeo);
1103         SAVE_INT32(ksi, mntinfo, mik_retrans);
1104         SAVE_UINT32(ksi, mntinfo, mik_acregmin);
1105         SAVE_UINT32(ksi, mntinfo, mik_acregmax);
1106         SAVE_UINT32(ksi, mntinfo, mik_acdirmin);
1107         SAVE_UINT32(ksi, mntinfo, mik_acdirmax);
1108         SAVE_UINT32_X(ksi, "lookup_srtt", mntinfo->mik_timers[0].srtt);
1109         SAVE_UINT32_X(ksi, "lookup_deviate", mntinfo->mik_timers[0].deviate);
1110         SAVE_UINT32_X(ksi, "lookup_rtxcur", mntinfo->mik_timers[0].rtxcur);
1111         SAVE_UINT32_X(ksi, "read_srtt", mntinfo->mik_timers[1].srtt);
1112         SAVE_UINT32_X(ksi, "read_deviate", mntinfo->mik_timers[1].deviate);
1113         SAVE_UINT32_X(ksi, "read_rtxcur", mntinfo->mik_timers[1].rtxcur);
1114         SAVE_UINT32_X(ksi, "write_srtt", mntinfo->mik_timers[2].srtt);
1115         SAVE_UINT32_X(ksi, "write_deviate", mntinfo->mik_timers[2].deviate);
1116         SAVE_UINT32_X(ksi, "write_rtxcur", mntinfo->mik_timers[2].rtxcur);
1117         SAVE_UINT32(ksi, mntinfo, mik_noresponse);
1118         SAVE_UINT32(ksi, mntinfo, mik_failover);
1119         SAVE_UINT32(ksi, mntinfo, mik_remap);
1120         SAVE_STRING(ksi, mntinfo, mik_curserver);
1121 }

1123 #ifdef __sparc
1124 static void
1125 save_sfmmu_global_stat(kstat_t *kp, ks_instance_t *ksi)
1126 {
1127         struct sfmmu_global_stat *sfmmug =
1128             (struct sfmmu_global_stat *)(kp->ks_data);

1130         assert(kp->ks_data_size == sizeof (struct sfmmu_global_stat));

1132         SAVE_INT32(ksi, sfmmug, sf_tsb_exceptions);
1133         SAVE_INT32(ksi, sfmmug, sf_tsb_raise_exception);
1134         SAVE_INT32(ksi, sfmmug, sf_pagefaults);
1135         SAVE_INT32(ksi, sfmmug, sf_uhash_searches);
1136         SAVE_INT32(ksi, sfmmug, sf_uhash_links);
1137         SAVE_INT32(ksi, sfmmug, sf_khash_searches);
1138         SAVE_INT32(ksi, sfmmug, sf_khash_links);
1139         SAVE_INT32(ksi, sfmmug, sf_swapout);
1140         SAVE_INT32(ksi, sfmmug, sf_tsb_alloc);
1141         SAVE_INT32(ksi, sfmmug, sf_tsb_allocfail);
1142         SAVE_INT32(ksi, sfmmug, sf_tsb_sectsb_create);
1143         SAVE_INT32(ksi, sfmmug, sf_scd_1sttsb_alloc);
1144         SAVE_INT32(ksi, sfmmug, sf_scd_2ndtsb_alloc);
1145         SAVE_INT32(ksi, sfmmug, sf_scd_1sttsb_allocfail);
1146         SAVE_INT32(ksi, sfmmug, sf_scd_2ndtsb_allocfail);
1147         SAVE_INT32(ksi, sfmmug, sf_tteload8k);
1148         SAVE_INT32(ksi, sfmmug, sf_tteload64k);
1149         SAVE_INT32(ksi, sfmmug, sf_tteload512k);
1150         SAVE_INT32(ksi, sfmmug, sf_tteload4m);
1151         SAVE_INT32(ksi, sfmmug, sf_tteload32m);
1152         SAVE_INT32(ksi, sfmmug, sf_tteload256m);
1153         SAVE_INT32(ksi, sfmmug, sf_tsb_load8k);
1154         SAVE_INT32(ksi, sfmmug, sf_tsb_load4m);
1155         SAVE_INT32(ksi, sfmmug, sf_hblk_hit);
1156         SAVE_INT32(ksi, sfmmug, sf_hblk8_ncreate);
1157         SAVE_INT32(ksi, sfmmug, sf_hblk8_nalloc);
1158         SAVE_INT32(ksi, sfmmug, sf_hblk1_ncreate);
1159         SAVE_INT32(ksi, sfmmug, sf_hblk1_nalloc);
1160         SAVE_INT32(ksi, sfmmug, sf_hblk_slab_cnt);
1161         SAVE_INT32(ksi, sfmmug, sf_hblk_reserve_cnt);
```



```
1162         SAVE_INT32(ksi, sfmmug, sf_hblk_recurse_cnt);
1163         SAVE_INT32(ksi, sfmmug, sf_hblk_reserve_hit);
1164         SAVE_INT32(ksi, sfmmug, sf_get_free_success);
1165         SAVE_INT32(ksi, sfmmug, sf_get_free_throttle);
1166         SAVE_INT32(ksi, sfmmug, sf_get_free_fail);
1167         SAVE_INT32(ksi, sfmmug, sf_put_free_success);
1168         SAVE_INT32(ksi, sfmmug, sf_put_free_fail);
1169         SAVE_INT32(ksi, sfmmug, sf_pgcolor_conflict);
1170         SAVE_INT32(ksi, sfmmug, sf_uncache_conflict);
1171         SAVE_INT32(ksi, sfmmug, sf_unload_conflict);
1172         SAVE_INT32(ksi, sfmmug, sf_ism_uncache);
1173         SAVE_INT32(ksi, sfmmug, sf_ism_recache);
1174         SAVE_INT32(ksi, sfmmug, sf_recache);
1175         SAVE_INT32(ksi, sfmmug, sf_steal_count);
1176         SAVE_INT32(ksi, sfmmug, sf_pagesync);
1177         SAVE_INT32(ksi, sfmmug, sf_clrwrt);
1178         SAVE_INT32(ksi, sfmmug, sf_pagesync_invalid);
1179         SAVE_INT32(ksi, sfmmug, sf_kernel_xcalls);
1180         SAVE_INT32(ksi, sfmmug, sf_user_xcalls);
1181         SAVE_INT32(ksi, sfmmug, sf_tsb_grow);
1182         SAVE_INT32(ksi, sfmmug, sf_tsb_shrink);
1183         SAVE_INT32(ksi, sfmmug, sf_tsb_resize_failures);
1184         SAVE_INT32(ksi, sfmmug, sf_tsb_reloc);
1185         SAVE_INT32(ksi, sfmmug, sf_user_vtop);
1186         SAVE_INT32(ksi, sfmmug, sf_ctx_inv);
1187         SAVE_INT32(ksi, sfmmug, sf_tlb_reprog_pgsz);
1188         SAVE_INT32(ksi, sfmmug, sf_region_remap_demap);
1189         SAVE_INT32(ksi, sfmmug, sf_create_scd);
1190         SAVE_INT32(ksi, sfmmug, sf_join_scd);
1191         SAVE_INT32(ksi, sfmmug, sf_leave_scd);
1192         SAVE_INT32(ksi, sfmmug, sf_destroy_scd);
1193 }
1194 #endif

1196 #ifdef __sparc
1197 static void
1198 save_sfmmu_tsbsize_stat(kstat_t *kp, ks_instance_t *ksi)
1199 {
1200         struct sfmmu_tsbsize_stat *sfmmut;

1202         assert(kp->ks_data_size == sizeof (struct sfmmu_tsbsize_stat));
1203         sfmmut = (struct sfmmu_tsbsize_stat *)(kp->ks_data);

1205         SAVE_INT32(ksi, sfmmut, sf_tsbsz_8k);
1206         SAVE_INT32(ksi, sfmmut, sf_tsbsz_16k);
1207         SAVE_INT32(ksi, sfmmut, sf_tsbsz_32k);
1208         SAVE_INT32(ksi, sfmmut, sf_tsbsz_64k);
1209         SAVE_INT32(ksi, sfmmut, sf_tsbsz_128k);
1210         SAVE_INT32(ksi, sfmmut, sf_tsbsz_256k);
1211         SAVE_INT32(ksi, sfmmut, sf_tsbsz_512k);
1212         SAVE_INT32(ksi, sfmmut, sf_tsbsz_1m);
1213         SAVE_INT32(ksi, sfmmut, sf_tsbsz_2m);
1214         SAVE_INT32(ksi, sfmmut, sf_tsbsz_4m);
1215 }
1216 #endif

1218 #ifdef __sparc
1219 static void
1220 save_simmstat(kstat_t *kp, ks_instance_t *ksi)
1221 {
1222         uchar_t *simmstat;
1223         char    *simm_buf;
1224         char    *list = NULL;
1225         int     i;

1227         assert(kp->ks_data_size == sizeof (uchar_t) * SIMM_COUNT);
```

```
1229         for (i = 0, simmstat = (uchar_t *)(kp->ks_data); i < SIMM_COUNT - 1;
1230             i++, simmstat++) {
1231                 if (list == NULL) {
1232                         (void) asprintf(&simm_buf, "%d,", *simmstat);
1233                 } else {
1234                         (void) asprintf(&simm_buf, "%s%d,", list, *simmstat);
1235                         free(list);
1236                 }
1237                 list = simm_buf;
1238         }

1240         (void) asprintf(&simm_buf, "%s%d", list, *simmstat);
1241         SAVE_STRING_X(ksi, "status", simm_buf);
1242         free(list);
1243         free(simm_buf);
1244 }
1245 #endif

1247 #ifdef __sparc
1248 /*
1249  * Helper function for save_temperature().
1250  */
1251 static char *
1252 short_array_to_string(short *shortp, int len)
1253 {
1254         char    *list = NULL;
1255         char    *list_buf;

1257         for (; len > 1; len--, shortp++) {
1258                 if (list == NULL) {
1259                         (void) asprintf(&list_buf, "%hd,", *shortp);
1260                 } else {
1261                         (void) asprintf(&list_buf, "%s%hd,", list, *shortp);
1262                         free(list);
1263                 }
1264                 list = list_buf;
1265         }

1267         (void) asprintf(&list_buf, "%s%hd", list, *shortp);
1268         free(list);
1269         return (list_buf);
1270 }

1272 static void
1273 save_temperature(kstat_t *kp, ks_instance_t *ksi)
1274 {
1275         struct temp_stats *temps = (struct temp_stats *)(kp->ks_data);
1276         char    *buf;

1278         assert(kp->ks_data_size == sizeof (struct temp_stats));

1280         SAVE_UINT32(ksi, temps, index);

1282         buf = short_array_to_string(temps->l1, L1_SZ);
1283         SAVE_STRING_X(ksi, "l1", buf);
1284         free(buf);

1286         buf = short_array_to_string(temps->l2, L2_SZ);
1287         SAVE_STRING_X(ksi, "l2", buf);
1288         free(buf);

1290         buf = short_array_to_string(temps->l3, L3_SZ);
1291         SAVE_STRING_X(ksi, "l3", buf);
1292         free(buf);
```



```
1294         buf = short_array_to_string(temps->l4, L4_SZ);
1295         SAVE_STRING_X(ksi, "l4", buf);
1296         free(buf);

1298         buf = short_array_to_string(temps->l5, L5_SZ);
1299         SAVE_STRING_X(ksi, "l5", buf);
1300         free(buf);

1302         SAVE_INT32(ksi, temps, max);
1303         SAVE_INT32(ksi, temps, min);
1304         SAVE_INT32(ksi, temps, state);
1305         SAVE_INT32(ksi, temps, temp_cnt);
1306         SAVE_INT32(ksi, temps, shutdown_cnt);
1307         SAVE_INT32(ksi, temps, version);
1308         SAVE_INT32(ksi, temps, trend);
1309         SAVE_INT32(ksi, temps, override);
1310 }
1311 #endif

1313 #ifdef __sparc
1314 static void
1315 save_temp_over(kstat_t *kp, ks_instance_t *ksi)
1316 {
1317         short   *sh = (short *)(kp->ks_data);
1318         char    *value;

1320         assert(kp->ks_data_size == sizeof (short));

1322         (void) asprintf(&value, "%hu", *sh);
1323         SAVE_STRING_X(ksi, "override", value);
1324         free(value);
1325 }
1326 #endif

1328 #ifdef __sparc
1329 static void
1330 save_ps_shadow(kstat_t *kp, ks_instance_t *ksi)
1331 {
1332         uchar_t *uchar = (uchar_t *)(kp->ks_data);

1334         assert(kp->ks_data_size == SYS_PS_COUNT);

1336         SAVE_CHAR_X(ksi, "core_0", *uchar++);
1337         SAVE_CHAR_X(ksi, "core_1", *uchar++);
1338         SAVE_CHAR_X(ksi, "core_2", *uchar++);
1339         SAVE_CHAR_X(ksi, "core_3", *uchar++);
1340         SAVE_CHAR_X(ksi, "core_4", *uchar++);
1341         SAVE_CHAR_X(ksi, "core_5", *uchar++);
1342         SAVE_CHAR_X(ksi, "core_6", *uchar++);
1343         SAVE_CHAR_X(ksi, "core_7", *uchar++);
1344         SAVE_CHAR_X(ksi, "pps_0", *uchar++);
1345         SAVE_CHAR_X(ksi, "clk_33", *uchar++);
1346         SAVE_CHAR_X(ksi, "clk_50", *uchar++);
1347         SAVE_CHAR_X(ksi, "v5_p", *uchar++);
1348         SAVE_CHAR_X(ksi, "v12_p", *uchar++);
1349         SAVE_CHAR_X(ksi, "v5_aux", *uchar++);
1350         SAVE_CHAR_X(ksi, "v5_p_pch", *uchar++);
1351         SAVE_CHAR_X(ksi, "v12_p_pch", *uchar++);
1352         SAVE_CHAR_X(ksi, "v3_pch", *uchar++);
1353         SAVE_CHAR_X(ksi, "v5_pch", *uchar++);
1354         SAVE_CHAR_X(ksi, "p_fan", *uchar++);
1355 }
1356 #endif

1358 #ifdef __sparc
1359 static void
```

```
save_fault_list(kstat_t *kp, ks_instance_t *ksi)
{
        struct ft_list *fault;
        char    name[KSTAT_STRLEN + 7];
        int     i;

        for (i = 1, fault = (struct ft_list *)(kp->ks_data);
            i <= 999999 && i <= kp->ks_data_size / sizeof (struct ft_list);
            i++, fault++) {
                (void) snprintf(name, sizeof (name), "unit_%d", i);
                SAVE_INT32_X(ksi, name, fault->unit);
                (void) snprintf(name, sizeof (name), "type_%d", i);
                SAVE_INT32_X(ksi, name, fault->type);
                (void) snprintf(name, sizeof (name), "fclass_%d", i);
                SAVE_INT32_X(ksi, name, fault->fclass);
                (void) snprintf(name, sizeof (name), "create_time_%d", i);
                SAVE_HRTIME_X(ksi, name, fault->create_time);
                (void) snprintf(name, sizeof (name), "msg_%d", i);
                SAVE_STRING_X(ksi, name, fault->msg);
        }
}
#endif

static void
save_named(kstat_t *kp, ks_instance_t *ksi)
{
        kstat_named_t *knp;
        int     n;

        for (n = kp->ks_ndata, knp = KSTAT_NAMED_PTR(kp); n > 0; n--, knp++) {
                switch (knp->data_type) {
                case KSTAT_DATA_CHAR:
                        nvpair_insert(ksi, knp->name,
                            (ks_value_t *)&knp->value, KSTAT_DATA_CHAR);
                        break;
                case KSTAT_DATA_INT32:
                        nvpair_insert(ksi, knp->name,
                            (ks_value_t *)&knp->value, KSTAT_DATA_INT32);
                        break;
                case KSTAT_DATA_UINT32:
                        nvpair_insert(ksi, knp->name,
                            (ks_value_t *)&knp->value, KSTAT_DATA_UINT32);
                        break;
                case KSTAT_DATA_INT64:
                        nvpair_insert(ksi, knp->name,
                            (ks_value_t *)&knp->value, KSTAT_DATA_INT64);
                        break;
                case KSTAT_DATA_UINT64:
                        nvpair_insert(ksi, knp->name,
                            (ks_value_t *)&knp->value, KSTAT_DATA_UINT64);
                        break;
                case KSTAT_DATA_STRING:
                        SAVE_STRING_X(ksi, knp->name, KSTAT_NAMED_STR_PTR(knp));
                        break;
                default:
                        assert(B_FALSE); /* Invalid data type */
                        break;
                }
        }
}

static void
save_intr(kstat_t *kp, ks_instance_t *ksi)
{
        kstat_intr_t *intr = KSTAT_INTR_PTR(kp);
        char    *intr_names[] = {"hard", "soft", "watchdog", "spurious",
```



```
            "multiple_service"};
        int     n;

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

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

        SAVE_UINT64(ksi, ksio, nread);
        SAVE_UINT64(ksi, ksio, nwritten);
        SAVE_UINT32(ksi, ksio, reads);
        SAVE_UINT32(ksi, ksio, writes);
        SAVE_HRTIME(ksi, ksio, wtime);
        SAVE_HRTIME(ksi, ksio, wlentime);
        SAVE_HRTIME(ksi, ksio, wlastupdate);
        SAVE_HRTIME(ksi, ksio, rtime);
        SAVE_HRTIME(ksi, ksio, rlentime);
        SAVE_HRTIME(ksi, ksio, rlastupdate);
        SAVE_UINT32(ksi, ksio, wcnt);
        SAVE_UINT32(ksi, ksio, rcnt);
}

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

        SAVE_STRING(ksi, ktimer, name);
        SAVE_UINT64(ksi, ktimer, num_events);
        SAVE_HRTIME(ksi, ktimer, elapsed_time);
        SAVE_HRTIME(ksi, ktimer, min_time);
        SAVE_HRTIME(ksi, ktimer, max_time);
        SAVE_HRTIME(ksi, ktimer, start_time);
        SAVE_HRTIME(ksi, ktimer, stop_time);
}
```